AV Foundation Framework Reference

Audio & Video

2010-07-13

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Simultaneously published in the United States and Canada.

Contents

AV Foundation Framework Reference

Framework	/System/Library/Frameworks/AVFoundation.framework
Header file directories	/System/Library/Frameworks/AVFoundation.framework/Headers
Declared in	AVAnimation.h AVAsset.h AVAssetExportSession.h AVAssetImageGenerator.h AVAssetTrack.h AVAssetTrackSegment.h AVAsynchronousKeyValueLoading.h AVAudioMix.h AVAudioPlayer.h AVAudioRecorder.h AVAudioSession.h AVAudioSettings.h AVCaptureDevice.h AVCaptureInput.h AVCaptureOutput.h AVCaptureSession.h AVCaptureVideoPreviewLayer.h AVComposition.h AVCompositionTrack.h AVCompositionTrackSegment.h AVError.h AVMediaFormat.h AVMetadataFormat.h AVMetadataItem.h AVPlayer.h AVPlayerItem.h AVPlayerItemTrack.h AVPlayerLayer.h AVSynchronizedLayer.h AVTime.h AVUtilities.h AVVideoComposition.h AVVideoSettings.h

Introduction

The AV Foundation framework provides an Objective-C interface for managing and playing audio-visual media in your iOS application.

Concurrent Programming with AV Foundation

Callouts from AV Foundation—invocations of blocks, key-value observers, or notification handlers—are not guaranteed to be made on any particular thread or queue. Instead, AV Foundation invokes these handlers on threads or queues on which it performs its internal tasks. You are responsible for testing whether the thread or queue on which a handler is invoked is appropriate for the tasks you want to perform. If it's not (for example, if you want to update the user interface and the callout is not on the main thread), you must redirect the execution of your tasks to a safe thread or queue that you recognize, or that you create for the purpose.

If you're writing a multithreaded application, you can use the `NSThread` method `isMainThread` or `[[NSThread currentThread] isEqual:<#A stored thread reference#>]` to testing whether the invocation thread is a thread you expect to perform your work on. You can redirect messages to appropriate threads using methods such as `performSelectorOnMainThread:withObject:waitUntilDone:` and `performSelector:onThread:withObject:waitUntilDone:modes:`. You could also use `dispatch_async` to "bounce" to your blocks on an appropriate queue, either the main queue for UI tasks or a queue you have up for concurrent operations. For more about concurrent operations, see *Concurrency Programming Guide*; for more about blocks, see *Blocks Programming Topics*.

Classes

AVAsset Class Reference

Inherits from	NSObject
Conforms to	NSCopying AVAsynchronousKeyValueLoading NSObject (NSObject)
Framework	/System/Library/Frameworks/AVFoundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	AVAsset.h AVVideoComposition.h

Overview

`AVAsset` is an abstract class to represent timed audiovisual media such as videos and sounds. Each asset contains a collection of tracks that are intended to be presented or processed together, each of a uniform media type, including but not limited to audio, video, text, closed captions, and subtitles.

An `AVAsset` object defines the collective properties of the tracks that comprise the asset. (You can access the instances of `AVAssetTrack` representing tracks of the collection, so you can examine each of these independently if you need to.) You often instantiate an asset using a concrete subclass of `AVAsset`; for example, you can initialize an instance of `AVURLAsset` using an URL that refers to an audiovisual media file, such as a QuickTime movie file or an MP3 files (amongst other types). You can also instantiate an asset using other concrete subclasses that extend the basic model for audiovisual media in useful ways, as `AVComposition` does for temporal editing. To assemble audiovisual constructs from one or more source assets, you can insert assets into instances of `AVMutableComposition`.

Inspecting and Loading Asset Data

Because of the nature of timed audiovisual media, successful initialization of an asset does not necessarily mean that all its data, and the values for its keys are immediately available. Instead, the asset will wait to load data until an operation is performed on it (for example, directly invoking any relevant `AVAsset` methods, playback via an `AVPlayerItem` object, export using `AVAssetExportSession`, and so on). You can request the value of any key at any time, and its value will be returned synchronously, however the calling thread may be blocked until the request can be satisfied. To avoid blocking, you ask for the values for particular keys to be loaded and to be notified when their values become available using `loadValuesAsynchronouslyForKeys:completionHandler:` (page 284).

Playing an Asset

You play an asset using an `AVPlayer` object via an instance of `AVPlayerItem`:

- First you initialize an instance of `AVPlayerItem` using the asset (see `playerItemWithAsset:` (page 246) and `initWithAsset:` (page 247)).
- Next you use the item to set up the asset's presentation state (such as whether only a limited time range of the asset should be played, and so on).
- Finally, you pass the player item to an `AVPlayer` object—either by initializing a new player using `playerWithPlayerItem:` (page 229) or `initWithPlayerItem:` (page 232), or (if you have an existing player) using `replaceCurrentItemWithPlayerItem:` (page 234).

Again, though, you must consider that when you create the asset it may not be ready for immediate playback. To ensure it can be played as soon as you associate it with a player, you can initialize an asset then ask for an observable property such as preferred volume (`preferredVolume` (page 22)) to be loaded. When the preferred volume value is available, the asset is ready to play and you can add it to the player.

Subclassing Notes

It is not currently possible to subclass `AVAsset` to handle streaming protocols or file formats that are not supported by the framework.

Tasks

Loading Data

- `cancelLoading` (page 23)
  Cancels the loading of all values for all observers.

Accessing Metadata

`commonMetadata` (page 20) *property*
An array of metadata items for each common metadata key for which a value is available. (read-only)

`availableMetadataFormats` (page 20) *property*
An array of strings, each representing a metadata format that's available to the asset. (read-only)

- `metadataForFormat:` (page 23)
  Returns an array of `AVMetadataItem` objects, one for each metadata item in the container of the specified format

`lyrics` (page 21) *property*
The lyrics of the asset suitable for the current locale. (read-only)

Accessing Tracks

`tracks` (page 22) *property*
The tracks contained by the asset. (read-only)

- `trackWithTrackID:` (page 25)
Returns the track with a specified track ID.

- `tracksWithMediaCharacteristic:` (page 24)
Returns an array of `AVAssetTrack` objects of the asset that present media with a specified characteristic.

- `tracksWithMediaType:` (page 24)
Returns an array of the asset tracks of the asset that present media of a specified type.

AVAssetVideoCompositionUtility

- `unusedTrackID` (page 25)
Returns a track ID for the asset.

Accessing Common Metadata

`duration` (page 20) *property*
The duration of the asset. (read-only)

`providesPreciseDurationAndTiming` (page 22) *property*
Indicates whether the asset provides precise timing. (read-only)

Preferred Asset Attributes

`naturalSize` (page 21) *property*
The encoded or authored size of the visual portion of the asset. (read-only)

`preferredRate` (page 21) *property*
The natural rate at which the asset is to be played. (read-only)

`preferredTransform` (page 21) *property*
The preferred transform to apply to the visual content of the asset for presentation or processing. (read-only)

`preferredVolume` (page 22) *property*
The preferred volume at which the audible media of asset is to be played. (read-only)

Properties

For more about Objective-C properties, see “Properties” in *The Objective-C Programming Language*.

availableMetadataFormats

An array of strings, each representing a metadata format that's available to the asset. (read-only)

```
@property(nonatomic, readonly) NSArray *availableMetadataFormats
```

Discussion
Metadata formats may include ID3, iTunes metadata, and so on. For more details, see `AVMetadataItem`.

Availability
Available in iOS 4.0 and later.

Declared In
`AVAsset.h`

commonMetadata

An array of metadata items for each common metadata key for which a value is available. (read-only)

```
@property(nonatomic, readonly) NSArray *commonMetadata
```

Discussion
The value is an array of `AVMetadataItem` objects, one for each common metadata key for which a value is available. You can filter the array by locale using `metadataItemsFromArray:withLocale:` (page 182) (`AVMetadataItem`) or by key using `metadataItemsFromArray:withKey:keySpace:` (page 182) (`AVMetadataItem`).

Availability
Available in iOS 4.0 and later.

Declared In
`AVAsset.h`

duration

The duration of the asset. (read-only)

```
@property(nonatomic, readonly) CMTime duration
```

Discussion
If `providesPreciseDurationAndTiming` (page 22) is `NO`, a best-available estimate of the duration is returned. You can set the degree of precision required for timing-related properties at initialization time for assets initialized with URLs (see `AVURLAssetPreferPreciseDurationAndTimingKey` in `AVURLAsset`).

Availability
Available in iOS 4.0 and later.

Declared In
`AVAsset.h`

lyrics

The lyrics of the asset suitable for the current locale. (read-only)

```
@property(nonatomic, readonly) NSString *lyrics
```

Availability
Available in iOS 4.0 and later.

Declared In
`AVAsset.h`

naturalSize

The encoded or authored size of the visual portion of the asset. (read-only)

```
@property(nonatomic, readonly) CGSize naturalSize
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVAsset.h`

preferredRate

The natural rate at which the asset is to be played. (read-only)

```
@property(nonatomic, readonly) float preferredRate
```

Discussion
This value is often, but not always, `1.0`.

Availability
Available in iOS 4.0 and later.

Declared In
`AVAsset.h`

preferredTransform

The preferred transform to apply to the visual content of the asset for presentation or processing. (read-only)

```
@property(nonatomic, readonly) CGAffineTransform preferredTransform
```

Discussion
The value is often, but not always, the identity transform.

Availability
Available in iOS 4.0 and later.

Declared In
AVAsset.h

preferredVolume

The preferred volume at which the audible media of asset is to be played. (read-only)

```
@property(nonatomic, readonly) float preferredVolume
```

Discussion
This value is often, but not always, 1.0.

Availability
Available in iOS 4.0 and later.

Declared In
AVAsset.h

providesPreciseDurationAndTiming

Indicates whether the asset provides precise timing. (read-only)

```
@property(nonatomic, readonly) BOOL providesPreciseDurationAndTiming
```

Discussion
You can set the degree of precision required for timing-related properties at initialization time for assets initialized with URLs (see AVURLAssetPreferPreciseDurationAndTimingKey in AVURLAsset).

Availability
Available in iOS 4.0 and later.

See Also
@property duration (page 20)

Declared In
AVAsset.h

tracks

The tracks contained by the asset. (read-only)

```
@property(nonatomic, readonly) NSArray *tracks
```

Discussion
Tracks are instances of AVAssetTrack.

Availability
Available in iOS 4.0 and later.

See Also
- tracksWithMediaType: (page 24)
- tracksWithMediaCharacteristic: (page 24)

- trackWithTrackID: (page 25)

Declared In
AVAsset.h

Instance Methods

cancelLoading

Cancels the loading of all values for all observers.

```
- (void)cancelLoading
```

Discussion
Deallocation of an instance of the asset will implicitly invoke this method if any loading requests are still outstanding.

Availability
Available in iOS 4.0 and later.

Declared In
AVAsset.h

metadataForFormat:

Returns an array of AVMetadataItem objects, one for each metadata item in the container of the specified format

```
- (NSArray *)metadataForFormat:(NSString *)format
```

Parameters
format
The metadata format for which you want items.

Return Value
An array of AVMetadataItem objects, one for each metadata item in the container of the specified format, or nil if there is no metadata of the specified format.

Discussion
You can filter the array by locale using metadataItemsFromArray:withLocale: (page 182) (AVMetadataItem) or by key using metadataItemsFromArray:withKey:keySpace: (page 182) (AVMetadataItem).

Special Considerations
Becomes callable without blocking when availableMetadataFormats (page 20) has been loaded.

Availability
Available in iOS 4.0 and later.

Declared In
AVAsset.h

tracksWithMediaCharacteristic:

Returns an array of `AVAssetTrack` objects of the asset that present media with a specified characteristic.

```
- (NSArray *)tracksWithMediaCharacteristic:(NSString *)mediaCharacteristic
```

Parameters

mediaCharacteristic

The media characteristic according to which receiver filters its asset tracks.

For valid values, see `AVAssetTrack`.

Return Value

An array of `AVAssetTrack` objects that present media with *mediaCharacteristic*, or `nil` if no tracks with the specified characteristic are available.

Discussion

You can call this method without blocking when `tracks` (page 22) has been loaded.

Availability

Available in iOS 4.0 and later.

See Also

- `tracksWithMediaType:` (page 24)
- `trackWithTrackID:` (page 25)

`@property tracks` (page 22)

Declared In

`AVAsset.h`

tracksWithMediaType:

Returns an array of the asset tracks of the asset that present media of a specified type.

```
- (NSArray *)tracksWithMediaType:(NSString *)mediaType
```

Parameters

mediaType

The media type according to which the asset filters its tracks.

Media types are defined in `AVAssetTrack`.

Return Value

An array of `AVAssetTrack` objects of the asset that present media of *mediaType*.

Discussion

You can call this method without blocking when `tracks` (page 22) has been loaded.

Availability

Available in iOS 4.0 and later.

See Also

- `tracksWithMediaCharacteristic:` (page 24)
- `trackWithTrackID:` (page 25)

`@property tracks` (page 22)

Declared In
AVAsset.h

trackWithTrackID:

Returns the track with a specified track ID.

```
- (AVAssetTrack *)trackWithTrackID:(CMPersistentTrackID)trackID
```

Parameters

trackID

The trackID of the requested asset track.

Return Value
The track with track ID *trackID*, or `nil` if no track with the specified ID is available.

Discussion
You can call this method without blocking when `tracks` (page 22) has been loaded.

Availability
Available in iOS 4.0 and later.

See Also
- `tracksWithMediaType:` (page 24)
- `tracksWithMediaCharacteristic:` (page 24)

`@property tracks` (page 22)

Declared In
AVAsset.h

unusedTrackID

Returns a track ID for the asset.

```
- (CMPersistentTrackID)unusedTrackID
```

Availability
Available in iOS 4.0 and later.

Declared In
AVVideoComposition.h

AVAssetExportSession Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/AVFoundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	AVAssetExportSession.h

Overview

An `AVAssetExportSession` object transcodes the contents of an `AVAsset` source object to create an output of the form described by a specified export preset.

Tasks

Initializing a Session

`- initWithAsset:presetName:` (page 36)
Initialize an asset export session with a specified preset and sets the source to the contents of the asset.

Exporting

`- exportAsynchronouslyWithCompletionHandler:` (page 35)
Starts the asynchronous execution of an export session.

`- cancelExport` (page 35)
Cancels the execution of an export session.

`error` (page 29) *property*
Describes the error that occurred if the export status is `AVAssetExportSessionStatusFailed` or `AVAssetExportSessionStatusCancelled`. (read-only)

`maxDuration` (page 30) *property*
The maximum duration that is allowed for export. (read-only)

Export Status

`progress` (page 31) *property*
The progress of the export on a scale from 0 to 1. (read-only)

`status` (page 32) *property*
The status of the export session. (read-only)

Configuring Output

`outputURL` (page 31) *property*
The URL of the export session's output.

`supportedFileTypes` (page 32) *property*
The types of files the session can write. (read-only)

`outputFileType` (page 30) *property*
The type of file to be written by the session.

`fileLengthLimit` (page 29) *property*
The maximum number of bytes that the session is allowed to write to the output URL.

`timeRange` (page 33) *property*
The time range to be exported from the source.

`metadata` (page 30) *property*
The metadata to be written to the output file by the export session.

`audioMix` (page 29) *property*
Indicates whether non-default audio mixing is enabled for export, and supplies the parameters for audio mixing.

`shouldOptimizeForNetworkUse` (page 32) *property*
Indicates whether the movie should be optimized for network use.

`videoComposition` (page 33) *property*
Indicates whether video composition is enabled for export, and supplies the instructions for video composition.

Export Presets

`presetName` (page 31) *property*
The name of the preset with which the session was initialized. (read-only)

`+ allExportPresets` (page 33)
Returns all available export preset names.

`+ exportPresetsCompatibleWithAsset:` (page 34)
Returns the identifiers compatible with a given asset.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C Programming Language*.

audioMix

Indicates whether non-default audio mixing is enabled for export, and supplies the parameters for audio mixing.

```
@property(nonatomic, copy) AVAudioMix *audioMix
```

Discussion
You can observe this property using key-value observing.

Availability
Available in iOS 4.0 and later.

Declared In
`AVAssetExportSession.h`

error

Describes the error that occurred if the export status is `AVAssetExportSessionStatusFailed` or `AVAssetExportSessionStatusCancelled`. (read-only)

```
@property(nonatomic, readonly) NSError *error
```

Discussion

Availability
Available in iOS 4.0 and later.

See Also
`- exportAsynchronouslyWithCompletionHandler:` (page 35)
`@property status` (page 32)

Declared In
`AVAssetExportSession.h`

fileLengthLimit

The maximum number of bytes that the session is allowed to write to the output URL.

```
@property(nonatomic) long long fileLengthLimit
```

Discussion
The export will stop when the output reaches this size regardless of the duration of the source or the value of `timeRange` (page 33).

You can observe this property using key-value observing.

Availability
Available in iOS 4.0 and later.

Declared In
`AVAssetExportSession.h`

maxDuration

The maximum duration that is allowed for export. (read-only)

```
@property(nonatomic, readonly) CMTime maxDuration
```

Discussion
You can observe this property using key-value observing.

Availability
Available in iOS 4.0 and later.

Declared In
`AVAssetExportSession.h`

metadata

The metadata to be written to the output file by the export session.

```
@property(nonatomic, copy) NSArray *metadata
```

Discussion
The metadata is an array of `AVMetadataItem` objects.

If the value of this key is `nil`, any existing metadata in the exported asset will be translated as accurately as possible into the appropriate metadata key space for the output file and written to the output.

You can observe this property using key-value observing.

Availability
Available in iOS 4.0 and later.

Declared In
`AVAssetExportSession.h`

outputFileType

The type of file to be written by the session.

```
@property(nonatomic, copy) NSString *outputFileType
```

Discussion
If the session supports only a single type of file, you do not need to set this property.

You can observe this property using key-value observing.

Availability
Available in iOS 4.0 and later.

See Also
`@property supportedFileTypes` (page 32)
`@property outputURL` (page 31)

Declared In
AVAssetExportSession.h

outputURL

The URL of the export session's output.

```
@property(nonatomic, copy) NSURL *outputURL
```

Discussion
For sessions that support multiple file types, if you have not set `outputFileType` (page 30), `AVAssetExportSession` will attempt to write the type of file indicated by `outputURL`'s path extension.

You can observe this property using key-value observing.

Availability
Available in iOS 4.0 and later.

See Also
`@property outputFileType` (page 30)

Declared In
AVAssetExportSession.h

presetName

The name of the preset with which the session was initialized. (read-only)

```
@property(nonatomic, readonly) NSString *presetName
```

Discussion
For possible values, see "Export Preset Names for Device-Appropriate QuickTime Files" (page 37), "Export Preset Names for QuickTime Files of a Given Size" (page 38), AVAssetExportSessionStatusCancelled (page 37), "Export Preset Name for iTunes Audio" (page 39), and "Export Preset Name for Pass-Through" (page 39).

You can observe this property using key-value observing.

Availability
Available in iOS 4.0 and later.

See Also
`- initWithAsset:presetName:` (page 36)

Declared In
AVAssetExportSession.h

progress

The progress of the export on a scale from 0 to 1. (read-only)

```
@property(nonatomic, readonly) float progress
```

Discussion
A value of `0` means the export has not yet begun, `1` means the export is complete.

Availability
Available in iOS 4.0 and later.

Declared In
`AVAssetExportSession.h`

shouldOptimizeForNetworkUse

Indicates whether the movie should be optimized for network use.

```
@property(nonatomic) BOOL shouldOptimizeForNetworkUse
```

Discussion
You can observe this property using key-value observing.

Availability
Available in iOS 4.0 and later.

Declared In
`AVAssetExportSession.h`

status

The status of the export session. (read-only)

```
@property(nonatomic, readonly) AVAssetExportSessionStatus status
```

Discussion
For possible values, see "Session Status" (page 37).

You can observe this property using key-value observing.

Availability
Available in iOS 4.0 and later.

Declared In
`AVAssetExportSession.h`

supportedFileTypes

The types of files the session can write. (read-only)

```
@property(nonatomic, readonly) NSArray *supportedFileTypes
```

Discussion
The types of files the session can write are determined by the asset and and export preset with which the session was initialized.

You can observe this property using key-value observing.

Availability
Available in iOS 4.0 and later.

See Also
`@property outputFileType` (page 30)

Declared In
`AVAssetExportSession.h`

timeRange

The time range to be exported from the source.

```
@property(nonatomic) CMTimeRange timeRange
```

Discussion
The default time range of an export session is `kCMTimeZero` to `kCMTimePositiveInfinity`, meaning that (modulo a possible limit on file length) the full duration of the asset will be exported.

You can observe this property using key-value observing.

Availability
Available in iOS 4.0 and later.

Declared In
`AVAssetExportSession.h`

videoComposition

Indicates whether video composition is enabled for export, and supplies the instructions for video composition.

```
@property(nonatomic, copy) AVVideoComposition *videoComposition
```

Discussion
You can observe this property using key-value observing.

Availability
Available in iOS 4.0 and later.

Declared In
`AVAssetExportSession.h`

Class Methods

allExportPresets

Returns all available export preset names.

```
+ (NSArray *)allExportPresets
```

Return Value
An array containing a string constant for each of the available preset names.

For possible values, see "Export Preset Names for Device-Appropriate QuickTime Files" (page 37), "Export Preset Names for QuickTime Files of a Given Size" (page 38), AVAssetExportSessionStatusCancelled (page 37), "Export Preset Name for iTunes Audio" (page 39), and "Export Preset Name for Pass-Through" (page 39).

Discussion
Not all presets are compatible with all assets.

Availability
Available in iOS 4.0 and later.

See Also
`+ exportPresetsCompatibleWithAsset:` (page 34)

Declared In
`AVAssetExportSession.h`

exportPresetsCompatibleWithAsset:

Returns the identifiers compatible with a given asset.

```
+ (NSArray *)exportPresetsCompatibleWithAsset:(AVAsset *)asset
```

Parameters

asset

An asset that is ready to be exported.

Return Value
An array containing strings representing the identifiers compatible with *asset*.

The array is a complete list of the valid identifiers that can be used with `initWithAsset:presetName:` (page 36) with the specified asset. For possible values, see "Export Preset Names for Device-Appropriate QuickTime Files" (page 37), "Export Preset Names for QuickTime Files of a Given Size" (page 38), AVAssetExportSessionStatusCancelled (page 37), "Export Preset Name for iTunes Audio" (page 39), and "Export Preset Name for Pass-Through" (page 39).

Discussion
Not all export presets are compatible with all assets (for example, a video-only asset is not compatible with an audio-only preset). This method returns only the identifiers for presets that will be compatible with the given asset.

In order to ensure that the setup and running of an export operation will succeed using a given preset, you should not make significant changes to the asset (such as adding or deleting tracks) between retrieving compatible identifiers and performing the export operation.

Availability
Available in iOS 4.0 and later.

See Also
`+ allExportPresets` (page 33)

Declared In
`AVAssetExportSession.h`

Instance Methods

cancelExport

Cancels the execution of an export session.

```
- (void)cancelExport
```

Discussion
You can invoke this method when the export is running.

Availability
Available in iOS 4.0 and later.

Declared In
`AVAssetExportSession.h`

exportAsynchronouslyWithCompletionHandler:

Starts the asynchronous execution of an export session.

```
- (void)exportAsynchronouslyWithCompletionHandler:(void (^)(void))handler
```

Parameters
handler
A block that is invoked when writing is complete or in the event of writing failure.

Discussion
This method starts an asynchronous export operation and returns immediately. `status` (page 32) signals the terminal state of the export session, and if a failure occurs, `error` (page 29) describes the problem.

If internal preparation for export fails, *handler* is invoked synchronously. The handler may also be called asynchronously, after the method returns, in the following cases:

1. If a failure occurs during the export, including failures of loading, re-encoding, or writing media data to the output.
2. If `cancelExport` (page 35) is invoked.
3. After the export session succeeds, having completely written its output to the `outputURL` (page 31).

Availability
Available in iOS 4.0 and later.

See Also
- `cancelExport` (page 35)
- `@property status` (page 32)

```
@property error (page 29)
```

Declared In
AVAssetExportSession.h

initWithAsset:presetName:

Initialize an asset export session with a specified preset and sets the source to the contents of the asset.

```
- (id)initWithAsset:(AVAsset *)asset presetName:(NSString *)presetName
```

Parameters

asset

The asset you want to export.

presetName

A string constant specifying the name of the preset template for the export.

For possible values, see "Export Preset Names for Device-Appropriate QuickTime Files" (page 37), "Export Preset Names for QuickTime Files of a Given Size" (page 38), AVAssetExportSessionStatusCancelled (page 37), "Export Preset Name for iTunes Audio" (page 39), and "Export Preset Name for Pass-Through" (page 39).

Return Value
An asset export session initialized to export *asset* using preset *presetName*.

Availability
Available in iOS 4.0 and later.

Declared In
AVAssetExportSession.h

Constants

AVAssetExportSessionStatus

A type to specify the session's status.

```
typedef NSInteger AVAssetExportSessionStatus;
```

Discussion
For possible values, see "Session Status" (page 37).

Availability
Available in iOS 4.0 and later.

Declared In
AVAssetExportSession.h

Session Status

Constants to indicate the status of the session.

```
enum {
    AVAssetExportSessionStatusUnknown,
    AVAssetExportSessionStatusExporting,
    AVAssetExportSessionStatusCompleted,
    AVAssetExportSessionStatusFailed,
    AVAssetExportSessionStatusCancelled,
    AVAssetExportSessionStatusWaiting
};
```

Constants

`AVAssetExportSessionStatusUnknown`
Indicates that the status is unknown.
Available in iOS 4.0 and later.
Declared in `AVAssetExportSession.h`.

`AVAssetExportSessionStatusExporting`
Indicates that the export session is in progress.
Available in iOS 4.0 and later.
Declared in `AVAssetExportSession.h`.

`AVAssetExportSessionStatusCompleted`
Indicates that the export session completed successfully.
Available in iOS 4.0 and later.
Declared in `AVAssetExportSession.h`.

`AVAssetExportSessionStatusFailed`
Indicates that the export session failed.
Available in iOS 4.0 and later.
Declared in `AVAssetExportSession.h`.

`AVAssetExportSessionStatusCancelled`
Indicates that the export session was cancelled.
Available in iOS 4.0 and later.
Declared in `AVAssetExportSession.h`.

`AVAssetExportSessionStatusWaiting`
Indicates that the session is waiting to export more data.
Available in iOS 4.0 and later.
Declared in `AVAssetExportSession.h`.

Export Preset Names for Device-Appropriate QuickTime Files

You use these export options to produce QuickTime .mov files with video size appropriate to the current device.

```
NSString *const AVAssetExportPresetLowQuality;
NSString *const AVAssetExportPresetMediumQuality;
NSString *const AVAssetExportPresetHighestQuality;
```

Constants

`AVAssetExportPresetLowQuality`

Specifies a low quality QuickTime file.

Available in iOS 4.0 and later.

Declared in `AVAssetExportSession.h`.

`AVAssetExportPresetMediumQuality`

Specifies a medium quality QuickTime file.

Available in iOS 4.0 and later.

Declared in `AVAssetExportSession.h`.

`AVAssetExportPresetHighestQuality`

Specifies a high quality QuickTime file.

Available in iOS 4.0 and later.

Declared in `AVAssetExportSession.h`.

Discussion

The export will not scale the video up from a smaller size. Video is compressed using H.264; audio is compressed using AAC.

See also AVAssetExportSessionStatusCancelled (page 37).

Export Preset Names for QuickTime Files of a Given Size

You use these export options to produce QuickTime .mov files with a specified video size.

```
NSString *const AVAssetExportPreset640x480;
NSString *const AVAssetExportPreset960x540;
NSString *const AVAssetExportPreset1280x720;
NSString *const AVAssetExportPreset1920x1080;
```

Constants

`AVAssetExportPreset640x480`

Specifies output at 640x480 pixels.

Available in iOS 4.0 and later.

Declared in `AVAssetExportSession.h`.

`AVAssetExportPreset960x540`

Specifies output at 960x540 pixels.

Available in iOS 4.0 and later.

Declared in `AVAssetExportSession.h`.

`AVAssetExportPreset1280x720`

Specifies output at 1280x720 pixels.

Available in iOS 4.0 and later.

Declared in `AVAssetExportSession.h`.

`AVAssetExportPreset1920x1080`

Specifies output at 1920x1080 pixels.

Discussion
The export will not scale the video up from a smaller size. Video is compressed using H.264; audio is compressed using AAC. Some devices cannot support some sizes.

Export Preset Name for iTunes Audio

You use this export option to produce an audio-only .m4a file with appropriate iTunes gapless playback data.

```
NSString *const AVAssetExportPresetAppleM4A;
```

Constants
`AVAssetExportPresetAppleM4A`
Specifies an audio-only .m4a file with appropriate iTunes gapless playback data.
Available in iOS 4.0 and later.
Declared in `AVAssetExportSession.h`.

Export Preset Name for Pass-Through

You use this export option to let all tracks pass through.

```
NSString *const AVAssetExportPresetPassthrough;
```

Constants
`AVAssetExportPresetPassthrough`
Specifies that all tracks pass through, unless it is not possible.
Available in iOS 4.0 and later.
Declared in `AVAssetExportSession.h`.

Discussion
This option does not show up in the `allExportPresets` (page 33) and `exportPresetsCompatibleWithAsset:` (page 34) methods.

AVAssetImageGenerator Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/AVFoundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	AVAssetImageGenerator.h

Overview

An `AVAssetImageGenerator` object provides thumbnail or preview images of assets independently of playback.

`AVAssetImageGenerator` uses the default enabled video track(s) to generate images. Generating a single image in isolation can require the decoding of a large number of video frames with complex interdependencies. If you require a series of images, you can achieve far greater efficiency using the asynchronous method, `copyCGImageAtTime:actualTime:error:` (page 45), which employs decoding efficiencies similar to those used during playback.

You create an asset generator using `initWithAsset:` (page 46) or `assetImageGeneratorWithAsset:` (page 44). These methods may succeed even if the asset possesses no visual tracks at the time of initialization. You can test whether an asset has any tracks with the visual characteristic using `tracksWithMediaCharacteristic:` (page 24) (`AVAsset`).

Assets that represent mutable compositions or mutable movies may gain visual tracks after initialization of an associated image generator.

Tasks

Creating a Generator

`- initWithAsset:` (page 46)
Initializes an image generator for use with a specified asset.

`+ assetImageGeneratorWithAsset:` (page 44)
Returns an image generator for use with a specified asset.

Generating Images

- `copyCGImageAtTime:actualTime:error:` (page 45)
  Returns a CGImage for the asset at or near a specified time.
- `generateCGImagesAsynchronouslyForTimes:completionHandler:` (page 45)
  Creates a series of CGImage objects for an asset at or near specified times.
- `cancelAllCGImageGeneration` (page 44)
  Cancels all pending image generation requests.

Generation Behavior

`apertureMode` (page 42) *property*
Specifies the aperture mode for the generated image.

`appliesPreferredTrackTransform` (page 42) *property*
Specifies whether to apply the track matrix (or matrices) when extracting an image from the asset.

`maximumSize` (page 43) *property*
Specifies the maximum dimensions for generated image.

`videoComposition` (page 43) *property*
The video composition to use when extracting images from assets with multiple video tracks.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C Programming Language*.

apertureMode

Specifies the aperture mode for the generated image.

```
@property(nonatomic, copy) NSString *apertureMode
```

Discussion
The default value is `AVAssetImageGeneratorApertureModeCleanAperture` (page 47).

Availability
Available in iOS 4.0 and later.

Declared In
`AVAssetImageGenerator.h`

appliesPreferredTrackTransform

Specifies whether to apply the track matrix (or matrices) when extracting an image from the asset.

```
@property(nonatomic) BOOL appliesPreferredTrackTransform
```

Discussion
The default is `NO`. `AVAssetImageGenerator` only supports rotation by 90, 180, or 270 degrees.

This property is ignored if you set a value for the `videoComposition` (page 43) property.

Availability
Available in iOS 4.0 and later.

See Also
`preferredTransform`
`@property videoComposition` (page 43)

Declared In
`AVAssetImageGenerator.h`

maximumSize

Specifies the maximum dimensions for generated image.

```
@property(nonatomic) CGSize maximumSize
```

Discussion
The default value is `CGSizeZero`, which specifies the asset's unscaled dimensions.

`AVAssetImageGenerator` scales images such that they fit within the defined bounding box. Images are never scaled up. The aspect ratio of the scaled image is defined by the `apertureMode` (page 42) property.

Availability
Available in iOS 4.0 and later.

Declared In
`AVAssetImageGenerator.h`

videoComposition

The video composition to use when extracting images from assets with multiple video tracks.

```
@property(nonatomic, copy) AVVideoComposition *videoComposition
```

Discussion
If no video composition is specified, only the first enabled video track will be used. If a video composition is specified, the `appliesPreferredTrackTransform` (page 42) property is ignored.

Availability
Available in iOS 4.0 and later.

See Also
`@property appliesPreferredTrackTransform` (page 42)

Declared In
`AVAssetImageGenerator.h`

Class Methods

assetImageGeneratorWithAsset:

Returns an image generator for use with a specified asset.

```
+ (AVAssetImageGenerator *)assetImageGeneratorWithAsset:(AVAsset *)asset
```

Parameters

asset

The asset from which images will be extracted.

Return Value

An image generator for use with *asset*.

Discussion

This method may succeed even if the asset possesses no visual tracks at the time of initialization.

Availability

Available in iOS 4.0 and later.

See Also

`tracksWithMediaCharacteristic:` (page 24)

Declared In

`AVAssetImageGenerator.h`

Instance Methods

cancelAllCGImageGeneration

Cancels all pending image generation requests.

```
- (void)cancelAllCGImageGeneration
```

Discussion

This method calls the handler block with `AVAssetImageGeneratorCancelled` (page 48) for each image time in every previous invocation of `generateCGImagesAsynchronouslyForTimes:completionHandler:` (page 45) for which images have not yet been supplied.

Availability

Available in iOS 4.0 and later.

See Also

- `copyCGImageAtTime:actualTime:error:` (page 45)
- `generateCGImagesAsynchronouslyForTimes:completionHandler:` (page 45)

Declared In

`AVAssetImageGenerator.h`

copyCGImageAtTime:actualTime:error:

Returns a CGImage for the asset at or near a specified time.

```
- (CGImageRef)copyCGImageAtTime:(CMTime)requestedTime
    actualTime:(CMTime *)actualTime
    error:(NSError **)outError
```

Parameters

requestedTime

The time at which the image of the asset is to be created.

actualTime

Upon return, contains the time at which the image was actually generated.

If you are not interested in this information, pass `NULL`.

outError

If an error occurs, upon return contains an `NSError` object that describes the problem.

If you are not interested in this information, pass `NULL`.

Return Value
A CGImage for the asset at or near a specified time, or `NULL` if the image could not be created.

This method follows "The Create Rule" in *Memory Management Programming Guide for Core Foundation*.

Discussion
This method returns the image synchronously.

Availability
Available in iOS 4.0 and later.

See Also
`- generateCGImagesAsynchronouslyForTimes:completionHandler:` (page 45)

Declared In
`AVAssetImageGenerator.h`

generateCGImagesAsynchronouslyForTimes:completionHandler:

Creates a series of CGImage objects for an asset at or near specified times.

```
- (void)generateCGImagesAsynchronouslyForTimes:(NSArray *)requestedTimes
    completionHandler:(AVAssetImageGeneratorCompletionHandler)handler
```

Parameters

requestedTimes

An array of `NSValue` objects, each containing a `CMTime`, specifying the asset times at which an image is requested.

handler

A block that is called when an image request is complete.

Discussion
This method uses an efficient "batch mode" to get images in time order.

The client receives exactly one handler callback for each requested time in `requestedTimes`. Changes to the generator's properties (snap behavior, maximum size, and so on) do not affect pending asynchronous image generation requests.

Availability
Available in iOS 4.0 and later.

See Also
- `cancelAllCGImageGeneration` (page 44)
- `copyCGImageAtTime:actualTime:error:` (page 45)

Declared In
`AVAssetImageGenerator.h`

initWithAsset:

Initializes an image generator for use with a specified asset.

```
- (id)initWithAsset:(AVAsset *)asset
```

Parameters

asset
The asset from which images will be extracted.

Return Value
An image generator initialized for use with *asset*.

Discussion
This method may succeed even if the asset possesses no visual tracks at the time of initialization.

Availability
Available in iOS 4.0 and later.

See Also
`tracksWithMediaCharacteristic:` (page 24)

Declared In
`AVAssetImageGenerator.h`

Constants

Aperture Modes

Constants to specify the aperture mode.

```
NSString *const AVAssetImageGeneratorApertureModeCleanAperture;
NSString *const AVAssetImageGeneratorApertureModeProductionAperture;
NSString *const AVAssetImageGeneratorApertureModeEncodedPixels;
```

Constants

`AVAssetImageGeneratorApertureModeCleanAperture`

Both pixel aspect ratio and clean aperture will be applied..

Available in iOS 4.0 and later.

Declared in `AVAssetImageGenerator.h`.

`AVAssetImageGeneratorApertureModeProductionAperture`

Only pixel aspect ratio will be applied.

Available in iOS 4.0 and later.

Declared in `AVAssetImageGenerator.h`.

`AVAssetImageGeneratorApertureModeEncodedPixels`

Neither pixel aspect ratio nor clean aperture will be applied.

Available in iOS 4.0 and later.

Declared in `AVAssetImageGenerator.h`.

AVAssetImageGeneratorCompletionHandler

This type specifies the signature for the block invoked when `generateCGImagesAsynchronouslyForTimes:completionHandler:` (page 45) has completed.

```
typedef void (^AVAssetImageGeneratorCompletionHandler)(CMTime requestedTime,
CGImageRef image, CMTime actualTime, AVAssetImageGeneratorResult result, NSError
*error)
```

Discussion

The block takes five arguments:

requestedTime

The time for which you requested an image.

image

The image that was generated, or `NULL` if the image could not be generated.

This parameter follows "The Get Rule" in *Memory Management Programming Guide for Core Foundation*.

actualTime

The time at which the image was actually generated.

result

A result code indicating whether the image generation process succeeded, failed, or was cancelled.

error

If `result` is `AVAssetImageGeneratorFailed` (page 48), an error object that describes the problem.

Availability

Available in iOS 4.0 and later.

Declared In

`AVAssetImageGenerator.h`

AVAssetImageGeneratorResult

Constants to indicate the outcome of image generation.

```
{
    AVAssetImageGeneratorSucceeded,
    AVAssetImageGeneratorFailed,
    AVAssetImageGeneratorCancelled,
};
typedef NSInteger AVAssetImageGeneratorResult;
```

Constants

`AVAssetImageGeneratorSucceeded`

Indicates that generation succeeded.

Available in iOS 4.0 and later.

Declared in `AVAssetImageGenerator.h`.

`AVAssetImageGeneratorFailed`

Indicates that generation failed.

Available in iOS 4.0 and later.

Declared in `AVAssetImageGenerator.h`.

`AVAssetImageGeneratorCancelled`

Indicates that generation was cancelled.

Available in iOS 4.0 and later.

Declared in `AVAssetImageGenerator.h`.

Discussion

These constants are used in the block completion handler for `generateCGImagesAsynchronouslyForTimes:completionHandler:` (page 45).

AVAssetTrack Class Reference

Inherits from	NSObject
Conforms to	NSCopying AVAsynchronousKeyValueLoading NSObject (NSObject)
Framework	/System/Library/Frameworks/AVFoundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	AVAssetTrack.h

Overview

An `AVAssetTrack` object provides provides the track-level inspection interface for all assets.

`AVAssetTrack` adopts the `AVAsynchronousKeyValueLoading` protocol. You should use methods in the protocol to make sure you access a track's properties without blocking the current thread. To cancel load requests for all keys of `AVAssetTrack` you must message the parent `AVAsset` object (for example, `[track.asset cancelLoading]`).

Tasks

Basic Properties

`asset` (page 51) *property*
The asset of which the track is a part. (read-only)

`trackID` (page 57) *property*
The persistent unique identifier for this track of the asset. (read-only)

`mediaType` (page 54) *property*
The media type for the track. (read-only)

`- hasMediaCharacteristic:` (page 57)
Returns a Boolean value that indicates whether the track references media with the specified media characteristic.

`formatDescriptions` (page 53) *property*
The formats of media samples referenced by the track. (read-only)

`enabled` (page 52) *property*
Indicates whether the track is enabled according to state stored in its container or construct. (read-only)

`selfContained` (page 56) *property*
Indicates whether the track references sample data only within its storage container. (read-only)

`totalSampleDataLength` (page 57) *property*
The total number of bytes of sample data required by the track. (read-only)

Temporal Properties

`timeRange` (page 56) *property*
The time range of the track within the overall timeline of the asset. (read-only)

`naturalTimeScale` (page 54) *property*
A timescale in which time values for the track can be operated upon without extraneous numerical conversion. (read-only)

`estimatedDataRate` (page 52) *property*
The estimated data rate of the media data referenced by the track, in bits per second. (read-only)

Track Language Properties

`languageCode` (page 53) *property*
The language associated with the track, as an ISO 639-2/T language code. (read-only)

`extendedLanguageTag` (page 53) *property*
The language tag associated with the track, as an RFC 4646 language tag. (read-only)

Visual Characteristics

`naturalSize` (page 54) *property*
The natural dimensions of the media data referenced by the track. (read-only)

`preferredTransform` (page 55) *property*
The transform specified in the track's storage container as the preferred transformation of the visual media data for display purposes. (read-only)

Audible Characteristics

`preferredVolume` (page 55) *property*
The volume specified in the track's storage container as the preferred volume of the audible media data. (read-only)

Frame-Based Characteristics

`nominalFrameRate` (page 55) *property*
The frame rate of the track, in frames per second. (read-only)

Track Segments

`segments` (page 56) *property*
The time mappings from the track's media samples to the timeline of the track. (read-only)

- `segmentForTrackTime:` (page 59)
The track segment that corresponds to the specified track time.

- `samplePresentationTimeForTrackTime:` (page 58)
Maps the specified track time through the appropriate time mapping and returns the resulting sample presentation time.

Managing Metadata

`commonMetadata` (page 52) *property*
An array of `AVMetadataItem` objects for each common metadata key for which a value is available. (read-only)

- `metadataForFormat:` (page 58)
An array of metadata items, one for each metadata item in the container of the specified format.

`availableMetadataFormats` (page 51) *property*
An array containing the metadata formats available for the track. (read-only)

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C Programming Language*.

asset

The asset of which the track is a part. (read-only)

```
@property(nonatomic, readonly) AVAsset *asset
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVAssetTrack.h`

availableMetadataFormats

An array containing the metadata formats available for the track. (read-only)

```
@property(nonatomic, readonly) NSArray *availableMetadataFormats
```

Discussion
The array contains `NSString` objects, one for each metadata format that's available for the track (such as QuickTime user data). For possible values, see `AVMetadataItem`.

Availability
Available in iOS 4.0 and later.

Declared In
`AVAssetTrack.h`

commonMetadata

An array of `AVMetadataItem` objects for each common metadata key for which a value is available. (read-only)

```
@property(nonatomic, readonly) NSArray *commonMetadata
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVAssetTrack.h`

enabled

Indicates whether the track is enabled according to state stored in its container or construct. (read-only)

```
@property(nonatomic, readonly, getter=isEnabled) BOOL enabled
```

Discussion
You can change the presentation state using `AVPlayerItem`.

Availability
Available in iOS 4.0 and later.

Declared In
`AVAssetTrack.h`

estimatedDataRate

The estimated data rate of the media data referenced by the track, in bits per second. (read-only)

```
@property(nonatomic, readonly) float estimatedDataRate
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
AVAssetTrack.h

extendedLanguageTag

The language tag associated with the track, as an RFC 4646 language tag. (read-only)

```
@property(nonatomic, readonly) NSString *extendedLanguageTag
```

Discussion
The value may be `nil` if no language tag is indicated.

Availability
Available in iOS 4.0 and later.

See Also
`@property languageCode` (page 53)

Declared In
AVAssetTrack.h

formatDescriptions

The formats of media samples referenced by the track. (read-only)

```
@property(nonatomic, readonly) NSArray *formatDescriptions
```

Discussion
The array contains CMFormatDescriptions (see `CMFormatDescriptionRef`), each of which indicates the format of media samples referenced by the track. A track that presents uniform media (for example, encoded according to the same encoding settings) will provide an array with a count of 1.

Availability
Available in iOS 4.0 and later.

See Also
`@property mediaType` (page 54)
`- hasMediaCharacteristic:` (page 57)

Declared In
AVAssetTrack.h

languageCode

The language associated with the track, as an ISO 639-2/T language code. (read-only)

```
@property(nonatomic, readonly) NSString *languageCode
```

Discussion
The value may be `nil` if no language is indicated.

Availability
Available in iOS 4.0 and later.

See Also
`@property extendedLanguageTag` (page 53)

Declared In
`AVAssetTrack.h`

mediaType

The media type for the track. (read-only)

```
@property(nonatomic, readonly) NSString *mediaType
```

Discussion
For possible values, see "Media Types" in *AV Foundation Constants Reference*.

Availability
Available in iOS 4.0 and later.

See Also
`- hasMediaCharacteristic:` (page 57)
`@property formatDescriptions` (page 53)

Declared In
`AVAssetTrack.h`

naturalSize

The natural dimensions of the media data referenced by the track. (read-only)

```
@property(nonatomic, readonly) CGSize naturalSize
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVAssetTrack.h`

naturalTimeScale

A timescale in which time values for the track can be operated upon without extraneous numerical conversion. (read-only)

```
@property(nonatomic, readonly) CMTimeScale naturalTimeScale
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVAssetTrack.h`

nominalFrameRate

The frame rate of the track, in frames per second. (read-only)

```
@property(nonatomic, readonly) float nominalFrameRate
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVAssetTrack.h`

preferredTransform

The transform specified in the track's storage container as the preferred transformation of the visual media data for display purposes. (read-only)

```
@property(nonatomic, readonly) CGAffineTransform preferredTransform
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVAssetTrack.h`

preferredVolume

The volume specified in the track's storage container as the preferred volume of the audible media data. (read-only)

```
@property(nonatomic, readonly) float preferredVolume
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVAssetTrack.h`

segments

The time mappings from the track's media samples to the timeline of the track. (read-only)

```
@property(nonatomic, copy, readonly) NSArray *segments
```

Discussion
The array contains instances of `AVAssetTrackSegment`.

Empty edits (that is, time ranges for which no media data is available to be presented) have `source.start` and `source.duration` equal to `kCMTimeInvalid`.

Availability
Available in iOS 4.0 and later.

See Also
`- segmentForTrackTime:` (page 59)

Declared In
`AVAssetTrack.h`

selfContained

Indicates whether the track references sample data only within its storage container. (read-only)

```
@property(nonatomic, readonly, getter=isSelfContained) BOOL selfContained
```

Discussion
The value is `YES` if the track references sample data only within its storage container, otherwise it is `NO`.

Availability
Available in iOS 4.0 and later.

Declared In
`AVAssetTrack.h`

timeRange

The time range of the track within the overall timeline of the asset. (read-only)

```
@property(nonatomic, readonly) CMTimeRange timeRange
```

Discussion
A track with `CMTimeCompare(timeRange.start, kCMTimeZero) == 1` will initially present an empty time range.

Availability
Available in iOS 4.0 and later.

Declared In
`AVAssetTrack.h`

totalSampleDataLength

The total number of bytes of sample data required by the track. (read-only)

```
@property(nonatomic, readonly) long long totalSampleDataLength
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVAssetTrack.h`

trackID

The persistent unique identifier for this track of the asset. (read-only)

```
@property(nonatomic, readonly) CMPersistentTrackID trackID
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVAssetTrack.h`

Instance Methods

hasMediaCharacteristic:

Returns a Boolean value that indicates whether the track references media with the specified media characteristic.

```
- (BOOL)hasMediaCharacteristic:(NSString *)mediaCharacteristic
```

Parameters

mediaCharacteristic

The media characteristic of interest.

For possible values, see “Media Characteristics” in *AV Foundation Constants Reference*.

Return Value
`YES` if the track references media with the specified characteristic, otherwise `NO`.

Discussion

Availability
Available in iOS 4.0 and later.

See Also
`@property mediaType` (page 54)

```
@property formatDescriptions (page 53)
```

Declared In
AVAssetTrack.h

metadataForFormat:

An array of metadata items, one for each metadata item in the container of the specified format.

```
- (NSArray *)metadataForFormat:(NSString *)format
```

Parameters

format

The metadata format for which items are requested.

Return Value
An array of `AVMetadataItem` objects, one for each metadata item in the container of the format specified by *format*, or `nil` if there is no metadata of the specified format.

Discussion
You can call this method without blocking after `availableMetadataFormats` (page 51) has been loaded.

Availability
Available in iOS 4.0 and later.

Declared In
AVAssetTrack.h

samplePresentationTimeForTrackTime:

Maps the specified track time through the appropriate time mapping and returns the resulting sample presentation time.

```
- (CMTime)samplePresentationTimeForTrackTime:(CMTime)trackTime
```

Parameters

trackTime

The track time for which a sample presentation time is requested.

Return Value
The sample presentation time corresponding to *trackTime*; the value will be invalid if *trackTime* is out of range.

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
AVAssetTrack.h

segmentForTrackTime:

The track segment that corresponds to the specified track time.

```
- (AVAssetTrackSegment *)segmentForTrackTime:(CMTime)trackTime
```

Parameters

trackTime

The track time for which you want the segment.

Return Value

The track segment from the segments array that corresponds to *trackTime*, or `nil` if *trackTime* is out of range.

Discussion

Availability

Available in iOS 4.0 and later.

See Also

`@property segments` (page 56)

Declared In

`AVAssetTrack.h`

AVAssetTrackSegment Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/AVFoundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	AVAssetTrackSegment.h

Overview

An `AVAssetTrackSegment` object represents a segment of an `AVAssetTrack` object, comprising of a time mapping from the source to the asset track timeline.

Tasks

Properties

`timeMapping` (page 62) *property*
The time range of the track of the container file of the media presented by the segment. (read-only)

`empty` (page 61) *property*
Indicates whether the segment is an empty segment (read-only)

Properties

For more about Objective-C properties, see “Properties” in *The Objective-C Programming Language*.

empty

Indicates whether the segment is an empty segment (read-only)

```
@property(nonatomic, readonly, getter=isEmpty) BOOL empty
```

Discussion
`YES` if the segment is empty, otherwise `NO`.

Availability
Available in iOS 4.0 and later.

Declared In
`AVAssetTrackSegment.h`

timeMapping

The time range of the track of the container file of the media presented by the segment. (read-only)

```
@property(nonatomic, readonly) CMTimeMapping timeMapping
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVAssetTrackSegment.h`

AVAudioMix Class Reference

Inherits from	NSObject
Conforms to	NSCopying NSMutableCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/AVFoundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	AVAudioMix.h

Overview

An `AVAudioMix` object manages the input parameters for mixing audio tracks. It allows custom audio processing to be performed on audio tracks during playback or other operations.

Tasks

Input Parameters

`inputParameters` (page 63) *property*
The parameters for inputs to the mix (read-only)

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C Programming Language*.

inputParameters

The parameters for inputs to the mix (read-only)

```
@property(nonatomic, readonly, copy) NSArray *inputParameters
```

Discussion
The array contains instances of `AVAudioMixInputParameters`. Note that an instance of `AVAudioMixInputParameters` is not required for each audio track that contributes to the mix; audio for those without associated `AVAudioMixInputParameters` objects will be included in the mix, processed according to default behavior.

Availability
Available in iOS 4.0 and later.

Declared In
`AVAudioMix.h`

AVAudioMixInputParameters Class Reference

Inherits from	NSObject
Conforms to	NSCopying NSMutableCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/AVFoundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	AVAudioMix.h

Overview

An `AVAudioMixInputParameters` object represents the parameters that should be applied to an audio track when it is added to a mix. Audio volume is currently supported as a time-varying parameter. `AVAudioMixInputParameters` has a mutable subclass, `AVMutableAudioMixInputParameters`.

You use an instance `AVAudioMixInputParameters` to apply audio volume ramps for an input to an audio mix. Mix parameters are associated with audio tracks via the `trackID` (page 66) property.

Before the first time at which a volume is set, a volume of 1.0 used; after the last time for which a volume has been set, the last volume is used. Within the time range of a volume ramp, the volume is interpolated between the start volume and end volume of the ramp. For example, setting the volume to 1.0 at time 0 and also setting a volume ramp from a volume of 0.5 to 0.2 with a timeRange of [4.0, 5.0] results in an audio volume parameters that hold the volume constant at 1.0 from 0.0 sec to 4.0 sec, then cause it to jump to 0.5 and descend to 0.2 from 4.0 sec to 9.0 sec, holding constant at 0.2 thereafter.

Tasks

Track ID

`trackID` (page 66) *property*
The trackID of the audio track to which the parameters should be applied. (read-only)

Getting Volume Ramps

`- getVolumeRampForTime:startVolume:endVolume:timeRange:` (page 66)
Obtains the volume ramp that includes the specified time.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C Programming Language*.

trackID

The trackID of the audio track to which the parameters should be applied. (read-only)

```
@property(nonatomic, readonly) CMPersistentTrackID trackID
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVAudioMix.h`

Instance Methods

getVolumeRampForTime:startVolume:endVolume:timeRange:

Obtains the volume ramp that includes the specified time.

```
- (BOOL)getVolumeRampForTime:(CMTime)time startVolume:(float *)startVolume
    endVolume:(float *)endVolume timeRange:(CMTimeRange *)timeRange
```

Parameters

time
If a ramp with a time range that contains the specified time has been set, information about the effective ramp for that time is supplied. Otherwise, information about the first ramp that starts after the specified time is supplied.

startVolume
A pointer to a float to receive the starting volume value for the volume ramp.

This value may be `NULL`.

endVolume
A pointer to a float to receive the ending volume value for the volume ramp.

This value may be `NULL`.

timeRange

A pointer to a CMTimeRange to receive the time range of the volume ramp.

This value may be `NULL`.

Return Value

`YES` if the values were retrieved successfully, otherwise `NO`. Returns `NO` if *time* is beyond the duration of the last volume ramp that has been set.

Discussion

Availability

Available in iOS 4.0 and later.

Declared In

`AVAudioMix.h`

AVAudioPlayer Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/AVFoundation.framework
Availability	Available in iOS 2.2 and later.
Declared in	AVAudioPlayer.h
Related sample code	AddMusic

Overview

An instance of the `AVAudioPlayer` class, called an audio player, provides playback of audio data from a file or memory.

Apple recommends that you use this class for audio playback unless you are playing audio captured from a network stream or require very low I/O latency. For an overview of audio technologies, see *Getting Started with Audio & Video* and "Using Audio" in *Multimedia Programming Guide*.

Using an audio player you can:

- Play sounds of any duration
- Play sounds from files or memory buffers
- Loop sounds
- Play multiple sounds simultaneously, one sound per audio player, with precise synchronization
- Control relative playback level and stereo positioning for each sound you are playing
- Seek to a particular point in a sound file, which supports such application features as fast forward and rewind
- Obtain data you can use for playback-level metering

The `AVAudioPlayer` class lets you play sound in any audio format available in iOS. You implement a delegate to handle interruptions (such as an incoming phone call) and to update the user interface when a sound has finished playing. The delegate methods to use are described in *AVAudioPlayerDelegate Protocol Reference*.

To play, pause, or stop an audio player, call one of its playback control methods, described in "Configuring and Controlling Playback" (page 70).

This class uses the Objective-C declared properties feature for managing information about a sound—such as the playback point within the sound's timeline, and for accessing playback options—such as volume and looping. You also use a property (`playing` (page 75)) to test whether or not playback is in progress.

To configure an appropriate audio session for playback, refer to *AVAudioSession Class Reference* and *AVAudioSessionDelegate Protocol Reference*. To learn how your choice of file formats impacts the simultaneous playback of multiple sounds, refer to "iPhone Hardware and Software Audio Codecs" in *Multimedia Programming Guide*.

Tasks

Initializing an AVAudioPlayer Object

- `initWithContentsOfURL:error:` (page 77)
  Initializes and returns an audio player for playing a designated sound file.
- `initWithData:error:` (page 77)
  Initializes and returns an audio player for playing a designated memory buffer.

Configuring and Controlling Playback

- `play` (page 79)
  Plays a sound asynchronously.
- `playAtTime:` (page 80)
  Plays a sound asynchronously, starting at a specified point in the audio output device's timeline.
- `pause` (page 78)
  Pauses playback; sound remains ready to resume playback from where it left off.
- `stop` (page 81)
  Stops playback and undoes the setup needed for playback.
- `prepareToPlay` (page 81)
  Prepares the audio player for playback by preloading its buffers.

`playing` (page 75) *property*
A Boolean value that indicates whether the audio player is playing (`YES`) or not (`NO`). (read-only)

`volume` (page 76) *property*
The playback gain for the audio player, ranging from 0.0 through 1.0.

`pan` (page 74) *property*
The audio player's stereo pan position.

`numberOfLoops` (page 74) *property*
The number of times a sound will return to the beginning, upon reaching the end, to repeat playback.

`delegate` (page 72) *property*
The delegate object for the audio player.

`settings` (page 75) *property*
The audio player's settings dictionary, containing information about the sound associated with the player. (read-only)

Managing Information About a Sound

`numberOfChannels` (page 74) *property*
The number of audio channels in the sound associated with the audio player. (read-only)

`duration` (page 73) *property*
Returns the total duration, in seconds, of the sound associated with the audio player. (read-only)

`currentTime` (page 71) *property*
The playback point, in seconds, within the timeline of the sound associated with the audio player.

`deviceCurrentTime` (page 72) *property*
The time value, in seconds, of the audio output device. (read-only)

`url` (page 76) *property*
The URL for the sound associated with the audio player. (read-only)

`data` (page 72) *property*
The data object containing the sound associated with the audio player. (read-only)

Using Audio Level Metering

`meteringEnabled` (page 73) *property*
A Boolean value that indicates the audio-level metering on/off state for the audio player.

- `averagePowerForChannel:` (page 76)
Returns the average power for a given channel, in decibels, for the sound being played.

- `peakPowerForChannel:` (page 78)
Returns the peak power for a given channel, in decibels, for the sound being played.

- `updateMeters` (page 82)
Refreshes the average and peak power values for all channels of an audio player.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C Programming Language*.

currentTime

The playback point, in seconds, within the timeline of the sound associated with the audio player.

```
@property NSTimeInterval currentTime
```

Discussion
If the sound is playing, `currentTime` is the offset of the current playback position, measured in seconds from the start of the sound. If the sound is not playing, `currentTime` is the offset of where playing starts upon calling the `play` (page 79) method, measured in seconds from the start of the sound.

By setting this property you can seek to a specific point in a sound file or implement audio fast-forward and rewind functions.

Availability
Available in iOS 2.2 and later.

See Also
`@property deviceCurrentTime` (page 72)
`@property duration` (page 73)

Declared In
`AVAudioPlayer.h`

data

The data object containing the sound associated with the audio player. (read-only)

```
@property(readonly) NSData *data
```

Discussion
Returns `nil` if the audio player has no data (that is, if it was not initialized with an `NSData` object).

Availability
Available in iOS 2.2 and later.

See Also
`@property url` (page 76)

Declared In
`AVAudioPlayer.h`

delegate

The delegate object for the audio player.

```
@property(assign) id<AVAudioPlayerDelegate> delegate
```

Discussion
The object that you assign to be an audio player's delegate becomes the target of the notifications described in *AVAudioPlayerDelegate Protocol Reference*. These notifications let you respond to decoding errors, audio interruptions (such as an incoming phone call), and playback completion.

Availability
Available in iOS 2.2 and later.

Declared In
`AVAudioPlayer.h`

deviceCurrentTime

The time value, in seconds, of the audio output device. (read-only)

```
@property(readonly) NSTimeInterval deviceCurrentTime
```

Discussion
The value of this property increases monotonically while an audio player is playing or paused.

If more than one audio player is connected to the audio output device, device time continues incrementing as long as at least one of the players is playing or paused.

If the audio output device has no connected audio players that are either playing or paused, device time reverts to 0.

Use this property to indicate "now" when calling the `playAtTime:` (page 80) instance method. By configuring multiple audio players to play at a specified offset from `deviceCurrentTime`, you can perform precise synchronization—as described in the discussion for that method.

Availability
Available in iOS 4.0 and later.

See Also
`@property currentTime` (page 71)
`- playAtTime:` (page 80)

Declared In
`AVAudioPlayer.h`

duration

Returns the total duration, in seconds, of the sound associated with the audio player. (read-only)

```
@property(readonly) NSTimeInterval duration
```

Availability
Available in iOS 2.2 and later.

See Also
`@property currentTime` (page 71)

Declared In
`AVAudioPlayer.h`

meteringEnabled

A Boolean value that indicates the audio-level metering on/off state for the audio player.

```
@property(getter=isMeteringEnabled) BOOL meteringEnabled
```

Discussion
The default value for the `meteringEnabled` property is off (Boolean `NO`). Before using metering for an audio player, you need to enable it by setting this property to `YES`. If `player` is an audio player instance variable of your controller class, you enable metering as shown here:

```
[self.player setMeteringEnabled: YES];
```

Availability
Available in iOS 2.2 and later.

See Also
- `averagePowerForChannel:` (page 76)
- `peakPowerForChannel:` (page 78)
- `updateMeters` (page 82)

Declared In
`AVAudioPlayer.h`

numberOfChannels

The number of audio channels in the sound associated with the audio player. (read-only)

```
@property(readonly) NSUInteger numberOfChannels
```

Availability
Available in iOS 2.2 and later.

Declared In
`AVAudioPlayer.h`

numberOfLoops

The number of times a sound will return to the beginning, upon reaching the end, to repeat playback.

```
@property NSInteger numberOfLoops
```

Discussion
A value of `0`, which is the default, means to play the sound once. Set a positive integer value to specify the number of times to return to the start and play again. For example, specifying a value of `1` results in a total of two plays of the sound. Set any negative integer value to loop the sound indefinitely until you call the `stop` (page 81) method.

Availability
Available in iOS 2.2 and later.

Declared In
`AVAudioPlayer.h`

pan

The audio player's stereo pan position.

```
@property float pan
```

Discussion
By setting this property you can position a sound in the stereo field. A value of –1.0 is full left, 0.0 is center, and 1.0 is full right.

Availability
Available in iOS 4.0 and later.

Declared In
`AVAudioPlayer.h`

playing

A Boolean value that indicates whether the audio player is playing (`YES`) or not (`NO`). (read-only)

```
@property(readonly, getter=isPlaying) BOOL playing
```

Discussion
To find out when playback has stopped, use the `audioPlayerDidFinishPlaying:successfully:` (page 288) delegate method.

Important: Do not poll this property (that is, do not use it inside of a loop) in an attempt to discover when playback has stopped.

Availability
Available in iOS 2.2 and later.

Related Sample Code
AddMusic

Declared In
`AVAudioPlayer.h`

settings

The audio player's settings dictionary, containing information about the sound associated with the player. (read-only)

```
@property(readonly) NSDictionary *settings
```

Discussion
An audio player's settings dictionary contains keys for the following information about the player's associated sound:

- Channel layout (`AVChannelLayoutKey` (page 311))
- Encoder bit rate (`AVEncoderBitRateKey` (page 310))
- Audio data format (`AVFormatIDKey` (page 309))
- Channel count (`AVNumberOfChannelsKey` (page 309))
- Sample rate (`AVSampleRateKey` (page 309))

The settings keys are described in *AV Foundation Audio Settings Constants*.

Availability
Available in iOS 4.0 and later.

Declared In
`AVAudioPlayer.h`

url

The URL for the sound associated with the audio player. (read-only)

```
@property(readonly) NSURL *url
```

Discussion
Returns `nil` if the audio player was not initialized with a URL.

Availability
Available in iOS 2.2 and later.

See Also
`@property data` (page 72)

Declared In
`AVAudioPlayer.h`

volume

The playback gain for the audio player, ranging from 0.0 through 1.0.

```
@property float volume
```

Availability
Available in iOS 2.2 and later.

Declared In
`AVAudioPlayer.h`

Instance Methods

averagePowerForChannel:

Returns the average power for a given channel, in decibels, for the sound being played.

```
- (float)averagePowerForChannel:(NSUInteger)channelNumber
```

Parameters

channelNumber

The audio channel whose average power value you want to obtain. Channel numbers are zero-indexed. A monaural signal, or the left channel of a stereo signal, has channel number `0`.

Return Value
A floating-point representation, in decibels, of a given audio channel's current average power. A return value of 0 dB indicates full scale, or maximum power; a return value of -160 dB indicates minimum power (that is, near silence).

If the signal provided to the audio player exceeds ±full scale, then the return value may exceed 0 (that is, it may enter the positive range).

Discussion
To obtain a current average power value, you must call the `updateMeters` (page 82) method before calling this method.

Availability
Available in iOS 2.2 and later.

See Also
`@property meteringEnabled` (page 73)
`- peakPowerForChannel:` (page 78)

Declared In
`AVAudioPlayer.h`

initWithContentsOfURL:error:

Initializes and returns an audio player for playing a designated sound file.

```
- (id)initWithContentsOfURL:(NSURL *)url error:(NSError **)outError
```

Parameters

url

A URL identifying the sound file to play. The audio data must be in a format supported by Core Audio. See "Using Sound in iOS" in *iOS Application Programming Guide*.

outError

On success, contains `nil`. On failure, contains an error code.

Return Value
On success, an initialized `AVAudioPlayer` object. If `nil`, the *outError* parameter contains a code that describes the problem.

Availability
Available in iOS 2.2 and later.

See Also
`- initWithData:error:` (page 77)

Related Sample Code
AddMusic

Declared In
`AVAudioPlayer.h`

initWithData:error:

Initializes and returns an audio player for playing a designated memory buffer.

```
- (id)initWithData:(NSData *)data error:(NSError **)outError
```

Parameters

data

A block of data containing a sound to play. The audio data must be in a format supported by Core Audio. See "Using Sound in iOS" in *iOS Application Programming Guide*.

outError

On success, contains `nil`. On failure, contains an error code.

Return Value

On success, an initialized `AVAudioPlayer` object. If `nil`, the *outError* parameter contains a code that describes the problem.

Availability

Available in iOS 2.2 and later.

See Also

- `initWithContentsOfURL:error:` (page 77)

Declared In

`AVAudioPlayer.h`

pause

Pauses playback; sound remains ready to resume playback from where it left off.

```
- (void)pause
```

Discussion

Calling `pause` leaves the audio player prepared to play; it does not release the audio hardware that was acquired upon calling `play` or `prepareToPlay`.

Availability

Available in iOS 2.2 and later.

See Also

- `play` (page 79)
- `prepareToPlay` (page 81)
- `stop` (page 81)

Declared In

`AVAudioPlayer.h`

peakPowerForChannel:

Returns the peak power for a given channel, in decibels, for the sound being played.

```
- (float)peakPowerForChannel:(NSUInteger)channelNumber
```

Parameters

channelNumber

The audio channel whose peak power value you want to obtain. Channel numbers are zero-indexed. A monaural signal, or the left channel of a stereo signal, has channel number `0`.

Return Value
A floating-point representation, in decibels, of a given audio channel's current peak power. A return value of 0 dB indicates full scale, or maximum power; a return value of -160 dB indicates minimum power (that is, near silence).

If the signal provided to the audio player exceeds ±full scale, then the return value may exceed 0 (that is, it may enter the positive range).

Discussion
To obtain a current peak power value, you must call the `updateMeters` (page 82) method before calling this method.

Availability
Available in iOS 2.2 and later.

See Also
`@property meteringEnabled` (page 73)
`- averagePowerForChannel:` (page 76)

Declared In
`AVAudioPlayer.h`

play

Plays a sound asynchronously.

```
- (BOOL)play
```

Return Value
Returns `YES` on success, or `NO` on failure.

Discussion
Calling this method implicitly calls the `prepareToPlay` method if the audio player is not already prepared to play.

Availability
Available in iOS 2.2 and later.

See Also
`- pause` (page 78)
`- playAtTime:` (page 80)
`- prepareToPlay` (page 81)
`- stop` (page 81)

Related Sample Code
AddMusic

Declared In
`AVAudioPlayer.h`

playAtTime:

Plays a sound asynchronously, starting at a specified point in the audio output device's timeline.

```
- (BOOL)playAtTime:(NSTimeInterval)time
```

Parameters

time

The number of seconds to delay playback, relative to the audio output device's current time. For example, to start playback three seconds into the future from the time you call this method, use code like this:

```
NSTimeInterval playbackDelay = 3.0;             // must be ≥ 0
[myAudioPlayer playAtTime: myAudioPlayer.deviceCurrentTime + playbackDelay];
```

Important: The value that you provide to the *time* parameter must be greater than or equal to the value of the audio player's `deviceCurrentTime` (page 72) property.

Return Value

`YES` on success, or `NO` on failure.

Discussion

Use this method to precisely synchronize the playback of two or more `AVAudioPlayer` objects. This code snippet shows the recommended way to do this:

```
// Before calling this method, instantiate two AVAudioPlayer objects and
// assign each of them a sound.

- (void) startSynchronizedPlayback {

    NSTimeInterval shortStartDelay = 0.01;            // seconds
    NSTimeInterval now = player.deviceCurrentTime;

    [player       playAtTime: now + shortStartDelay];
    [secondPlayer playAtTime: now + shortStartDelay];

    // Here, update state and user interface for each player, as appropriate
}
```

To learn about the virtual audio output device's timeline, read the description for the `deviceCurrentTime` (page 72) property.

Calling this method implicitly calls the `prepareToPlay` method if the audio player is not already prepared to play.

Availability

Available in iOS 4.0 and later.

See Also

- `pause` (page 78)
- `play` (page 79)
- `prepareToPlay` (page 81)
- `stop` (page 81)

Declared In
AVAudioPlayer.h

prepareToPlay

Prepares the audio player for playback by preloading its buffers.

```
- (BOOL)prepareToPlay
```

Return Value
Returns YES on success, or NO on failure.

Discussion
Calling this method preloads buffers and acquires the audio hardware needed for playback, which minimizes the lag between calling the play method and the start of sound output.

Calling the stop method, or allowing a sound to finish playing, undoes this setup.

Availability
Available in iOS 2.2 and later.

See Also
- pause (page 78)
- play (page 79)
- stop (page 81)

Declared In
AVAudioPlayer.h

stop

Stops playback and undoes the setup needed for playback.

```
- (void)stop
```

Discussion
Calling this method, or allowing a sound to finish playing, undoes the setup performed upon calling the play or prepareToPlay methods.

The stop method does not reset the value of the currentTime (page 71) property to 0. In other words, if you call stop during playback and then call play, playback resumes at the point where it left off.

Availability
Available in iOS 2.2 and later.

See Also
- pause (page 78)
- play (page 79)
- prepareToPlay (page 81)

Declared In
AVAudioPlayer.h

updateMeters

Refreshes the average and peak power values for all channels of an audio player.

```
- (void)updateMeters
```

Discussion
To obtain current audio power values, you must call this method before calling `averagePowerForChannel:` (page 76) or `peakPowerForChannel:` (page 78).

Availability
Available in iOS 2.2 and later.

See Also
`@property meteringEnabled` (page 73)

Declared In
`AVAudioPlayer.h`

AVAudioRecorder Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/AVFoundation.framework
Availability	Available in iOS 3.0 and later.
Declared in	

Overview

An instance of the `AVAudioRecorder` class, called an audio recorder, provides audio recording capability in your application. Using an audio recorder you can:

- Record until the user stops the recording
- Record for a specified duration
- Pause and resume a recording
- Obtain input audio-level data that you can use to provide level metering

You can implement a delegate object for an audio recorder to respond to audio interruptions and audio decoding errors, and to the completion of a recording.

To configure a recording, including options such as bit depth, bit rate, and sample rate conversion quality, configure the audio recorder's `settings` (page 86) dictionary. Use the settings keys described in *AV Foundation Audio Settings Constants*.

To configure an appropriate audio session for recording, refer to *AVAudioSession Class Reference* and *AVAudioSessionDelegate Protocol Reference*.

Tasks

Initializing an AVAudioRecorder Object

`- initWithURL:settings:error:` (page 87)
Initializes and returns an audio recorder.

Configuring and Controlling Recording

- `prepareToRecord` (page 89)
  Creates an audio file and prepares the system for recording.
- `record` (page 89)
  Starts or resumes recording.
- `recordForDuration:` (page 89)
  Records for a specified duration of time.
- `pause` (page 88)
  Pauses a recording.
- `stop` (page 90)
  Stops recording and closes the audio file.

`delegate` (page 85) *property*
The delegate object for the audio recorder.

- `deleteRecording` (page 87)
  Deletes a recorded audio file.

Managing Information About a Recording

`recording` (page 85) *property*
A Boolean value that indicates whether the audio recorder is recording (`YES`), or not (`NO`).

`url` (page 86) *property*
The URL for the audio file associated with the audio recorder.

`currentTime` (page 85) *property*
The time, in seconds, since the beginning of the recording.

`settings` (page 86) *property*
The audio settings for the audio recorder.

Using Audio Level Metering

`meteringEnabled` (page 85) *property*
A Boolean value that indicates whether audio-level metering is enabled (`YES`), or not (`NO`).

- `updateMeters` (page 90)
  Refreshes the average and peak power values for all channels of an audio recorder.
- `peakPowerForChannel:` (page 88)
  Returns the peak power for a given channel, in decibels, for the sound being recorded.
- `averagePowerForChannel:` (page 86)
  Returns the average power for a given channel, in decibels, for the sound being recorded.

Properties

For more about Objective-C properties, see “Properties” in *The Objective-C Programming Language*.

currentTime

The time, in seconds, since the beginning of the recording.

```
@property (readonly) NSTimeInterval currentTime;
```

Discussion
When the audio recorder is stopped, calling this method returns a value of `0`.

Availability
Available in iOS 3.0 and later.

Declared In
`AVAudioRecorder.h`

delegate

The delegate object for the audio recorder.

```
@property (assign) id <AVAudioRecorderDelegate> delegate;
```

Discussion
For a description of the audio recorder delegate, see *AVAudioRecorderDelegate Protocol Reference*.

Availability
Available in iOS 3.0 and later.

Declared In
`AVAudioRecorder.h`

meteringEnabled

A Boolean value that indicates whether audio-level metering is enabled (`YES`), or not (`NO`).

```
@property (getter=isMeteringEnabled) BOOL meteringEnabled;
```

Discussion
By default, audio level metering is off for an audio recorder. Because metering uses computing resources, turn it on only if you intend to use it.

Availability
Available in iOS 3.0 and later.

Declared In
`AVAudioRecorder.h`

recording

A Boolean value that indicates whether the audio recorder is recording (`YES`), or not (`NO`).

```
@property (readonly, getter=isRecording) BOOL recording;
```

Availability
Available in iOS 3.0 and later.

Declared In
`AVAudioRecorder.h`

settings

The audio settings for the audio recorder.

```
@property (readonly) NSDictionary *settings;
```

Discussion
Audio recorder settings are in effect only after you explicitly call the `prepareToRecord` (page 89) method, or after you call it implicitly by starting recording. The audio settings keys are described in *AV Foundation Audio Settings Constants*.

Availability
Available in iOS 3.0 and later.

Declared In
`AVAudioRecorder.h`

url

The URL for the audio file associated with the audio recorder.

```
@property (readonly) NSURL *url;
```

Availability
Available in iOS 3.0 and later.

Declared In
`AVAudioRecorder.h`

Instance Methods

averagePowerForChannel:

Returns the average power for a given channel, in decibels, for the sound being recorded.

```
- (float)averagePowerForChannel:(NSUInteger)channelNumber
```

Parameters

channelNumber

The number of the channel that you want the average power value for.

Return Value
The current average power, in decibels, for the sound being recorded. A return value of 0 dB indicates full scale, or maximum power; a return value of -160 dB indicates minimum power (that is, near silence).

If the signal provided to the audio recorder exceeds ±full scale, then the return value may exceed 0 (that is, it may enter the positive range).

Discussion
To obtain a current average power value, you must call the `updateMeters` (page 90) method before calling this method.

Availability
Available in iOS 3.0 and later.

See Also
`@property meteringEnabled` (page 85)
`- peakPowerForChannel:` (page 88)

Declared In
`AVAudioRecorder.h`

deleteRecording

Deletes a recorded audio file.

```
- (BOOL)deleteRecording
```

Return Value
Returns `YES` on success, or `NO` on failure.

Discussion
The audio recorder must be stopped before you call this method.

Availability
Available in iOS 3.0 and later.

Declared In
`AVAudioRecorder.h`

initWithURL:settings:error:

Initializes and returns an audio recorder.

```
- (id)initWithURL:(NSURL *)url
    settings:(NSDictionary *)settings
    error:(NSError **)outError
```

Parameters

url

The file system location to record to. The file type to record to is inferred from the file extension included in this parameter's value.

settings
Settings for the recording session. For information on the settings available for an audio recorder, see *AV Foundation Audio Settings Constants*.

outError
On success, contains `nil`. On failure, contains an error code.

Return Value
On success, an initialized `AVAudioRecorder` object. If `nil`, the *outError* parameter contains a code that describes the problem.

Availability
Available in iOS 3.0 and later.

Declared In
`AVAudioRecorder.h`

pause

Pauses a recording.

```
- (void)pause
```

Discussion
Call `record` (page 89) to resume recording.

Availability
Available in iOS 3.0 and later.

Declared In
`AVAudioRecorder.h`

peakPowerForChannel:

Returns the peak power for a given channel, in decibels, for the sound being recorded.

```
- (float)peakPowerForChannel:(NSUInteger)channelNumber
```

Parameters
channelNumber
The number of the channel that you want the peak power value for.

Return Value
The current peak power, in decibels, for the sound being recorded. A return value of 0 dB indicates full scale, or maximum power; a return value of -160 dB indicates minimum power (that is, near silence).

If the signal provided to the audio recorder exceeds ±full scale, then the return value may exceed 0 (that is, it may enter the positive range).

Discussion
To obtain a current peak power value, call the `updateMeters` (page 90) method immediately before calling this method.

Availability
Available in iOS 3.0 and later.

See Also
- `averagePowerForChannel:` (page 86)
`@property meteringEnabled` (page 85)

Declared In
`AVAudioRecorder.h`

prepareToRecord

Creates an audio file and prepares the system for recording.

```
- (BOOL)prepareToRecord
```

Return Value
Returns `YES` on success, or `NO` on failure.

Discussion
Creates an audio file at the location specified by the *url* parameter in the `initWithURL:settings:error:` (page 87) method. If a file already exists at that location, this method overwrites it.

The preparation invoked by this method takes place automatically when you call `record` (page 89). Use `prepareToRecord` when you want recording to start as quickly as possible upon calling `record`.

Availability
Available in iOS 3.0 and later.

Declared In
`AVAudioRecorder.h`

record

Starts or resumes recording.

```
- (BOOL)record
```

Return Value
Returns `YES` on success, or `NO` on failure.

Discussion
Calling this method implicitly calls `prepareToRecord` (page 89), which creates (or erases) an audio file and prepares the system for recording.

Availability
Available in iOS 3.0 and later.

Declared In
`AVAudioRecorder.h`

recordForDuration:

Records for a specified duration of time.

```
- (BOOL)recordForDuration:(NSTimeInterval)duration
```

Parameters

duration

The maximum duration, in seconds, for the recording.

Return Value

Returns `YES` on success, or `NO` on failure.

Discussion

The recorder stops when the duration of recorded audio reaches the value in the *duration* parameter.

Calling this method implicitly calls `prepareToRecord` (page 89), which creates (or erases) an audio file and prepares the system for recording.

Availability

Available in iOS 3.0 and later.

Declared In

`AVAudioRecorder.h`

stop

Stops recording and closes the audio file.

```
- (void)stop
```

Availability

Available in iOS 3.0 and later.

Declared In

`AVAudioRecorder.h`

updateMeters

Refreshes the average and peak power values for all channels of an audio recorder.

```
- (void)updateMeters
```

Discussion

To obtain current audio power values, you must call this method before you call `averagePowerForChannel:` (page 86) or `peakPowerForChannel:` (page 88).

Availability

Available in iOS 3.0 and later.

See Also

`@property meteringEnabled` (page 85)

Declared In

`AVAudioRecorder.h`

AVAudioSession Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/AVFoundation.framework/
Availability	Available in iOS 3.0 and later.
Declared in	
Companion guide	Audio Session Programming Guide
Related sample code	AddMusic

Overview

An instance of the `AVAudioSession` class, called an audio session, is a singleton object that you employ to set the audio context for your application. Use this class to:

- Activate or deactivate your application's audio session
- Set the audio session category
- Specify your preferred audio hardware sample rate and I/O buffer duration

Starting with iOS 3.0, this class provides an Objective-C alternative to most features from the C-based Audio Session Services, described in *Audio Session Services Reference*. Certain audio session features, such as handling audio route changes, can be accessed only using Audio Session Services.

For more information on audio sessions, see *Audio Session Programming Guide*.

Tasks

Instantiating an Audio Session

`+ sharedInstance` (page 95)
Returns the singleton audio session.

Specifying a Delegate

`delegate` (page 94) *property*
Specifies the delegate object for the audio session.

Managing an Audio Session

`category` (page 92) *property*
The audio session category.

- `setActive:error:` (page 95)
Activates or deactivates your application's audio session.

- `setActive:withFlags:error:` (page 96)
Activates or deactivates your application's audio session; provides flags for use by other audio sessions.

- `setCategory:error:` (page 96)
Sets the audio session category.

Working with Audio Hardware

`currentHardwareInputNumberOfChannels` (page 93) *property*
The number of audio hardware input channels.

`currentHardwareOutputNumberOfChannels` (page 93) *property*
The number of audio hardware output channels.

`currentHardwareSampleRate` (page 93) *property*
The audio hardware sample rate, in hertz.

`inputIsAvailable` (page 94) *property*
A Boolean value that indicates whether audio input is available (`YES`), or not (`NO`).

`preferredHardwareSampleRate` (page 94) *property*
The preferred hardware sample rate, in hertz.

`preferredIOBufferDuration` (page 95) *property*
The preferred I/O buffer duration, in seconds.

- `setPreferredHardwareSampleRate:error:` (page 97)
Sets the preferred hardware sample rate for recording.

- `setPreferredIOBufferDuration:error:` (page 97)
Sets the preferred I/O buffer duration, in seconds, for recording.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C Programming Language*.

category

The audio session category.

```
@propertyy) NSString* category
```

Discussion
An audio session has one of the categories listed in "Audio Session Categories" (page 98). The default category is `AVAudioSessionCategorySoloAmbient` (page 98).

Availability
Available in iOS 3.0 and later.

See Also
`- setCategory:error:` (page 96)

Declared In
`AVAudioSession.h`

currentHardwareInputNumberOfChannels

The number of audio hardware input channels.

```
@propertyy) NSInteger currentHardwareInputNumberOfChannels
```

Availability
Available in iOS 3.0 and later.

Declared In
`AVAudioSession.h`

currentHardwareOutputNumberOfChannels

The number of audio hardware output channels.

```
@propertyy) NSInteger currentHardwareOutputNumberOfChannels
```

Availability
Available in iOS 3.0 and later.

Declared In
`AVAudioSession.h`

currentHardwareSampleRate

The audio hardware sample rate, in hertz.

```
@propertyy) double currentHardwareSampleRate
```

Discussion
Obtain the value of this property after activating your audio session. You can request a hardware sample rate

Availability
Available in iOS 3.0 and later.

See Also
- `setPreferredHardwareSampleRate:error:` (page 97)

Declared In
`AVAudioSession.h`

delegate

Specifies the delegate object for the audio session.

```
@property id<AVAudioSessionDelegate>delegate
```

Discussion
The delegate object must implement the protocol described in *AVAudioSessionDelegate Protocol Reference*.

Availability
Available in iOS 3.0 and later.

Declared In
`AVAudioSession.h`

inputIsAvailable

A Boolean value that indicates whether audio input is available (`YES`), or not (`NO`).

```
@propertyy) BOOL inputIsAvailable
```

Discussion
Use this method on launch to determine whether the device your application is running on supports audio input. To respond to a change in the availability of audio input, implement the `inputIsAvailableChanged:` delegate method.

Availability
Available in iOS 3.0 and later.

Declared In
`AVAudioSession.h`

preferredHardwareSampleRate

The preferred hardware sample rate, in hertz.

```
@propertyy) double preferredHardwareSampleRate
```

Availability
Available in iOS 3.0 and later.

See Also
- `setPreferredHardwareSampleRate:error:` (page 97)

Declared In
`AVAudioSession.h`

preferredIOBufferDuration

The preferred I/O buffer duration, in seconds.

```
@propertyy) NSTimeInterval preferredIOBufferDuration
```

Availability
Available in iOS 3.0 and later.

See Also
- setPreferredIOBufferDuration:error: (page 97)

Declared In
AVAudioSession.h

Class Methods

sharedInstance

Returns the singleton audio session.

```
+ (id)sharedInstance
```

Availability
Available in iOS 3.0 and later.

Related Sample Code
AddMusic

Declared In
AVAudioSession.h

Instance Methods

setActive:error:

Activates or deactivates your application's audio session.

```
- (BOOL)setActive:(BOOL)beActive
    error:(NSError**)outError
```

Parameters

beActive
Use YES to activate your application's audio session or NO to deactivate it.

outError
Contains nil if the activation/deactivation call was successful; otherwise, contains an error code.

Return Value
Returns YES on success or NO on failure.

Discussion
If another application's active audio session has higher priority than your application, and that other audio session does not allow mixing, attempting to activate your audio session may fail.

Availability
Available in iOS 3.0 and later.

Related Sample Code
AddMusic

Declared In
`AVAudioSession.h`

setActive:withFlags:error:

Activates or deactivates your application's audio session; provides flags for use by other audio sessions.

```
- (BOOL)setActive:(BOOL)beActive
    withFlags:(NSInteger)flags
    error:(NSError**)outError
```

Parameters

beActive
Use `YES` to activate your application's audio session or `NO` to deactivate it.

flags
A bitmapped value containing one or more flags from the "Activation Flags" (page 99) enumeration.

outError
Contains `nil` if the activation/deactivation call was successful; otherwise, contains an error code.

Return Value
Returns `YES` on success or `NO` on failure.

Discussion
If another application's active audio session has higher priority than your application, and that other audio session does not allow mixing, attempting to activate your audio session may fail.

Availability
Available in iOS 4.0 and later.

Declared In
`AVAudioSession.h`

setCategory:error:

Sets the audio session category.

```
- (BOOL)setCategory:(NSString*)theCategory
    error:(NSError**)outError
```

Parameters

theCategory

The audio session category you want to apply to the audio session. See "Audio Session Categories" (page 98).

outError

Contains `nil` if the category was set; otherwise, contains an error code.

Return Value

Returns `YES` on success or `NO` on failure.

Availability

Available in iOS 3.0 and later.

See Also

`@property category` (page 92)

Related Sample Code

AddMusic

Declared In

`AVAudioSession.h`

setPreferredHardwareSampleRate:error:

Sets the preferred hardware sample rate for recording.

```
- (BOOL)setPreferredHardwareSampleRate:(double)sampleRate
    error:(NSError**)outError
```

Parameters

sampleRate

The hardware sample rate you want to use.

outError

On success, contains `nil`. On failure, contains an error code.

Return Value

Returns `YES` on success or `NO` on failure.

Availability

Available in iOS 3.0 and later.

See Also

`@property preferredHardwareSampleRate` (page 94)

Declared In

`AVAudioSession.h`

setPreferredIOBufferDuration:error:

Sets the preferred I/O buffer duration, in seconds, for recording.

```
- (BOOL)setPreferredIOBufferDuration:(NSTimeInterval)duration
    error:(NSError**)outError
```

Parameters

duration

The I/O buffer duration, in seconds, that you want to use.

outError

On success, contains `nil`. On failure, contains an error code.

Return Value

Returns `YES` on success or `NO` on failure.

Availability

Available in iOS 3.0 and later.

See Also

`@property preferredIOBufferDuration` (page 95)

Declared In

`AVAudioSession.h`

Constants

Audio Session Categories

Category identifiers for audio sessions, used as values for the `setCategory:error:` (page 96) method.

```
NSString *const AVAudioSessionCategoryAmbient;
NSString *const AVAudioSessionCategorySoloAmbient;
NSString *const AVAudioSessionCategoryPlayback;
NSString *const AVAudioSessionCategoryRecord;
NSString *const AVAudioSessionCategoryPlayAndRecord;
NSString *const AVAudioSessionCategoryAudioProcessing;
```

Constants

`AVAudioSessionCategoryAmbient`

For an application in which sound playback is nonprimary—that is, your application can be used successfully with the sound turned off.

This category is also appropriate for "play along" style applications, such as a virtual piano that a user plays over iPod audio. When you use this category, other audio, such as from the iPod application, mixes with your audio. Your audio is silenced by screen locking and by the Ring/Silent switch.

Available in iOS 3.0 and later.

Declared in `AVAudioSession.h`.

`AVAudioSessionCategorySoloAmbient`

The default category; used unless you set a category with the `setCategory:error:` (page 96) method.

This category silences audio from other applications, such as the iPod. Your audio is silenced by screen locking and by the Ring/Silent switch.

Available in iOS 3.0 and later.

Declared in `AVAudioSession.h`.

`AVAudioSessionCategoryPlayback`
: For playing recorded music or other sounds that are central to the successful use of your application.

  This category silences audio from other applications, such as the iPod. You can, however, modify this category to allow mixing by using the `kAudioSessionProperty_OverrideCategoryMixWithOthers` property. Your audio continues with the Ring/Silent switch set to silent and with the screen locked.

  Available in iOS 3.0 and later.

  Declared in `AVAudioSession.h`.

`AVAudioSessionCategoryRecord`
: For recording audio; this category silences playback audio. Recording continues with the screen locked.

  Available in iOS 3.0 and later.

  Declared in `AVAudioSession.h`.

`AVAudioSessionCategoryPlayAndRecord`
: For recording and playback of audio—simultaneous or not—such as for a VOIP (voice over IP) application.

  This category silences audio from other applications, such as the iPod. You can, however, modify this category to allow mixing by using the `kAudioSessionProperty_OverrideCategoryMixWithOthers` property. Your audio continues with the Ring/Silent switch set to silent and with the screen locked.

  Available in iOS 3.0 and later.

  Declared in `AVAudioSession.h`.

`AVAudioSessionCategoryAudioProcessing`
: For using an audio hardware codec or signal processor while not playing or recording audio. Use this category, for example, when performing offline audio format conversion.

  This category disables playback (audio output) and disables recording (audio input).

  Audio processing does not normally continue when your application is in the background. However, when your application moves to the background, you can request additional time to complete processing. for more information, see "Understanding an Application's States and Transitions" in *iOS Application Programming Guide*.

  Available in iOS 3.1 and later.

  Declared in `AVAudioSession.h`.

Discussion
Each application running in iOS has a single audio session, which can be active or inactive. You can change your audio session's category while your program is running.

Use the `AVAudioSessionCategoryAmbient` category when you want your sounds to mix with other audio (such as from the iPod application). Use one of the other playback categories when you want iPod audio to be silenced when your session is active. For more information on audio session categories, see *Audio Session Programming Guide*.

Activation Flags

Flags that provide additional information about your application's audio intentions upon session activation or deactivation.

```
enum {
    AVAudioSessionSetActiveFlags_NotifyOthersOnDeactivation = 1
};
```

Constants

`AVAudioSessionSetActiveFlags_NotifyOthersOnDeactivation`

When passed in the *flags* parameter of the `setActive:withFlags:error:` (page 96) instance method, indicates that when your audio session deactivates, other audio sessions that had been interrupted by your session can return to their active state.

This flag is used only when deactivating your audio session; that is, when you pass a value of `NO` in the *beActive* parameter of the `setActive:withFlags:error:` (page 96) instance method.

Available in iOS 4.0 and later.

Declared in `AVAudioSession.h`.

Interruption Flags

Constants that indicate the state of the audio session following an interruption.

```
enum {
    AVAudioSessionInterruptionFlags_ShouldResume = 1
};
```

Constants

`AVAudioSessionInterruptionFlags_ShouldResume`

Indicates that your audio session is active and immediately ready to be used. Your application can resume the audio operation that was interrupted.

Look for this flag in the *flags* parameter when your audio session delegate's `endInterruptionWithFlags:` (page 297) method is invoked.

Available in iOS 4.0 and later.

Declared in `AVAudioSession.h`.

AVCaptureAudioDataOutput Class Reference

Inherits from	AVCaptureOutput : NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/AVFoundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	AVCaptureOutput.h

Overview

`AVCaptureAudioDataOutput` is a concrete sub-class of `AVCaptureOutput` that you use, via its delegate, to process audio sample buffers from the audio being captured.

Tasks

Managing the Delegate

`- setSampleBufferDelegate:queue:` (page 102)
Sets the sample buffer delegate and the queue on which callbacks should be invoked.

`sampleBufferDelegate` (page 102) *property*
The capture object's delegate.

`sampleBufferCallbackQueue` (page 101) *property*
The queue on which delegate callbacks are invoked (read-only)

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C Programming Language*.

sampleBufferCallbackQueue

The queue on which delegate callbacks are invoked (read-only)

```
@property(nonatomic, readonly) dispatch_queue_t sampleBufferCallbackQueue
```

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureOutput.h`

sampleBufferDelegate

The capture object's delegate.

```
@property(nonatomic, readonly) id<AVCaptureAudioDataOutputSampleBufferDelegate>
    sampleBufferDelegate
```

Discussion
You use the delegate to manage incoming data.

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureOutput.h`

Instance Methods

setSampleBufferDelegate:queue:

Sets the sample buffer delegate and the queue on which callbacks should be invoked.

```
- (void)setSampleBufferDelegate:(id < AVCaptureAudioDataOutputSampleBufferDelegate
    >)sampleBufferDelegate queue:(dispatch_queue_t)sampleBufferCallbackQueue
```

Parameters

`sampleBufferDelegate`

The sample buffer delegate.

`sampleBufferCallbackQueue`

The queue on which callbacks should be invoked.

If you pass NULL, the delegate uses `dispatch_get_current_queue`.

Discussion
The delegate is sent a message each time a buffer of audio data is available.

Special Considerations

This method uses `dispatch_retain` and `dispatch_release` to manage the queue.

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureOutput.h`

AVCaptureConnection Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/AVFoundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	AVCaptureSession.h

Overview

An `AVCaptureConnection` object represents a connection between a capture input and a capture output added to a capture session.

Capture inputs (instances of `AVCaptureInput`) have one or more input ports (instances of `AVCaptureInputPort`). Capture outputs (instances of `AVCaptureOutput`) can accept data from one or more sources (for example, an `AVCaptureMovieFileOutput` object accepts both video and audio data).

When an input or an output is added to a session, the session greedily forms connections between all the compatible capture inputs' ports and capture outputs. You use connections to enable or disable the flow of data from a given input or to a given output.

Tasks

Configuration

`enabled` (page 104) *property*
Indicates whether the connection is enabled.

`active` (page 104) *property*
Indicates whether the connection is active. (read-only)

`inputPorts` (page 105) *property*
The connection's input ports. (read-only)

`output` (page 105) *property*
The connection's output port. (read-only)

`audioChannels` (page 104) *property*
An array of `AVCaptureAudioChannel` objects. (read-only)

`videoMirrored` (page 106) *property*
Indicates whether the video is mirrored.

`supportsVideoMirroring` (page 105) *property*
Indicates whether the connection supports mirroring of the video. (read-only)

`videoOrientation` (page 106) *property*
Indicates the orientation of the video.

`supportsVideoOrientation` (page 106) *property*
Indicates whether the connection supports changing the orientation of the video. (read-only)

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C Programming Language*.

active

Indicates whether the connection is active. (read-only)

```
@property(nonatomic, readonly, getter=isActive) BOOL active
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureSession.h`

audioChannels

An array of `AVCaptureAudioChannel` objects. (read-only)

```
@property(nonatomic, readonly) NSArray *audioChannels
```

Discussion
This property is only applicable to connections involving audio.

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureSession.h`

enabled

Indicates whether the connection is enabled.

```
@property(nonatomic, getter=isEnabled) BOOL enabled
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureSession.h`

inputPorts

The connection's input ports. (read-only)

```
@property(nonatomic, readonly) NSArray *inputPorts
```

Discussion
Input ports are instances of `AVCaptureInputPort`.

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureSession.h`

output

The connection's output port. (read-only)

```
@property(nonatomic, readonly) AVCaptureOutput *output
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureSession.h`

supportsVideoMirroring

Indicates whether the connection supports mirroring of the video. (read-only)

```
@property(nonatomic, readonly, getter=isVideoMirroringSupported) BOOL
    supportsVideoMirroring
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureSession.h`

supportsVideoOrientation

Indicates whether the connection supports changing the orientation of the video. (read-only)

```
@property(nonatomic, readonly, getter=isVideoOrientationSupported) BOOL
    supportsVideoOrientation
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureSession.h`

videoMirrored

Indicates whether the video is mirrored.

```
@property(nonatomic, getter=isVideoMirrored) BOOL videoMirrored
```

Discussion
This property is only applicable to connections involving video.

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureSession.h`

videoOrientation

Indicates the orientation of the video.

```
@property(nonatomic) AVCaptureVideoOrientation videoOrientation
```

Discussion
This property is only applicable to connections involving video.

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureSession.h`

AVCaptureDevice Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/AVFoundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	AVCaptureDevice.h

Overview

An `AVCaptureDevice` object abstracts a physical capture device that provides input data (such as audio or video) to an `AVCaptureSession` object.

You can enumerate the available devices, query their capabilities, and be informed when devices come and go. If you find a suitable capture device, you create an `AVCaptureDeviceInput` object for the device, and add that input to a capture session.

To set properties on an a capture device (its focus mode, exposure mode, and so on), you must first acquire a lock on the device using `lockForConfiguration:` (page 121). You should only hold the device lock if you need settable device properties to remain unchanged. Holding the device lock unnecessarily may degrade capture quality in other applications sharing the device.

Tasks

Discovering Devices

`+ devices` (page 117)
Returns an array containing the available capture devices on the system.

`+ deviceWithUniqueID:` (page 117)
Returns the device with a given ID.

`+ defaultDeviceWithMediaType:` (page 116)
Returns the default device used to capture data of a given media type.

`+ devicesWithMediaType:` (page 117)
Returns an array containing the devices able to capture data of a given media type

Focus Settings

`focusMode` (page 112) *property*
The device's focus mode.

- `isFocusModeSupported:` (page 119)
Returns a Boolean value that indicates whether the given focus mode is supported.

`focusPointOfInterest` (page 113) *property*
The point of interest for focusing.

`focusPointOfInterestSupported` (page 113) *property*
Indicates whether the device supports a point of interest for focus. (read-only)

`adjustingFocus` (page 110) *property*
Indicates whether the device is currently adjusting its focus setting. (read-only)

Exposure Settings

`adjustingExposure` (page 109) *property*
Indicates whether the device is currently adjusting its exposure setting. (read-only)

`exposureMode` (page 111) *property*
The exposure mode for the device.

- `isExposureModeSupported:` (page 118)
Returns a Boolean value that indicates whether the given exposure mode is supported.

`exposurePointOfInterest` (page 111) *property*
The point of interest for exposure.

`exposurePointOfInterestSupported` (page 112) *property*
Indicates whether the device supports a point of interest for exposure. (read-only)

Flash Settings

`hasFlash` (page 113) *property*
Indicates whether the capture device has a flash. (read-only)

`flashMode` (page 112) *property*
The current flash mode.

- `isFlashModeSupported:` (page 119)
Returns a Boolean value that indicates whether the given flash mode is supported.

White Balance Settings

- `isWhiteBalanceModeSupported:` (page 120)
Returns a Boolean value that indicates whether the given white balance mode is supported.

`whiteBalanceMode` (page 116) *property*
The current white balance mode.

`adjustingWhiteBalance` (page 110) *property*
Indicates whether the devise is currently adjusting the white balance. (read-only)

Torch Mode Settings

`hasTorch` (page 114) *property*
A Boolean value that specifies whether the capture device has a torch. (read-only)

- `isTorchModeSupported:` (page 120)
Returns a Boolean value that indicates whether the given torch mode is supported.

`torchMode` (page 115) *property*
The current torch mode.

Device Characteristics

`connected` (page 111) *property*
Indicates whether the device is currently connected. (read-only)

`position` (page 115) *property*
(read-only)

- `hasMediaType:` (page 118)

`modelID` (page 115) *property*
(read-only)

`localizedName` (page 114) *property*
(read-only)

`uniqueID` (page 116) *property*
(read-only)

- `supportsAVCaptureSessionPreset:` (page 121)

Locking the Device

- `lockForConfiguration:` (page 121)
Attempts to acquire a lock on the capture device.

- `unlockForConfiguration` (page 121)
Relinquishes a lock on a device.

Properties

For more about Objective-C properties, see “Properties” in *The Objective-C Programming Language*.

adjustingExposure

Indicates whether the device is currently adjusting its exposure setting. (read-only)

```
@property(nonatomic, readonly, getter=isAdjustingExposure) BOOL adjustingExposure
```

Discussion

Availability
Available in iOS 4.0 and later.

See Also
`@property exposureMode` (page 111)
`@property exposurePointOfInterest` (page 111)

Declared In
`AVCaptureDevice.h`

adjustingFocus

Indicates whether the device is currently adjusting its focus setting. (read-only)

```
@property(nonatomic, readonly, getter=isAdjustingFocus) BOOL adjustingFocus
```

Discussion

Availability
Available in iOS 4.0 and later.

See Also
`@property focusPointOfInterestSupported` (page 113)
`@property focusPointOfInterest` (page 113)
`- isFocusModeSupported:` (page 119)

Declared In
`AVCaptureDevice.h`

adjustingWhiteBalance

Indicates whether the devise is currently adjusting the white balance. (read-only)

```
@property(nonatomic, readonly, getter=isAdjustingWhiteBalance) BOOL
    adjustingWhiteBalance
```

Discussion

Availability
Available in iOS 4.0 and later.

See Also
`- isWhiteBalanceModeSupported:` (page 120)
`@property whiteBalanceMode` (page 116)

Declared In
`AVCaptureDevice.h`

connected

Indicates whether the device is currently connected. (read-only)

```
@property(nonatomic, readonly, getter=isConnected) BOOL connected
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
AVCaptureDevice.h

exposureMode

The exposure mode for the device.

```
@property(nonatomic) AVCaptureExposureMode exposureMode
```

Discussion
See "Exposure Modes" (page 125) for possible values.

Availability
Available in iOS 4.0 and later.

See Also
- `isExposureModeSupported:` (page 118)
- `@property adjustingExposure` (page 109)
- `@property exposurePointOfInterest` (page 111)
- `lockForConfiguration:` (page 121)

Declared In
AVCaptureDevice.h

exposurePointOfInterest

The point of interest for exposure.

```
@property(nonatomic) CGPoint exposurePointOfInterest
```

Discussion

Availability
Available in iOS 4.0 and later.

See Also
- `@property adjustingExposure` (page 109)
- `@property exposurePointOfInterestSupported` (page 112)

Declared In
AVCaptureDevice.h

exposurePointOfInterestSupported

Indicates whether the device supports a point of interest for exposure. (read-only)

```
@property(nonatomic, readonly, getter=isExposurePointOfInterestSupported) BOOL
    exposurePointOfInterestSupported
```

Discussion

Availability
Available in iOS 4.0 and later.

See Also
`@property exposurePointOfInterest` (page 111)
`- isExposureModeSupported:` (page 118)
`@property exposureMode` (page 111)

Declared In
`AVCaptureDevice.h`

flashMode

The current flash mode.

```
@property(nonatomic) AVCaptureFlashMode flashMode
```

Discussion
See "Flash Modes" (page 123) for possible values.

Availability
Available in iOS 4.0 and later.

See Also
`@property hasFlash` (page 113)
`- isFlashModeSupported:` (page 119)
`- lockForConfiguration:` (page 121)

Declared In
`AVCaptureDevice.h`

focusMode

The device's focus mode.

```
@property(nonatomic) AVCaptureFocusMode focusMode
```

Discussion
See "Focus Modes" (page 124) for possible values.

Availability
Available in iOS 4.0 and later.

See Also
`@property focusPointOfInterestSupported` (page 113)
`@property focusPointOfInterest` (page 113)
`- isFocusModeSupported:` (page 119)
`- lockForConfiguration:` (page 121)

Declared In
`AVCaptureDevice.h`

focusPointOfInterest

The point of interest for focusing.

```
@property(nonatomic) CGPoint focusPointOfInterest
```

Discussion

Availability
Available in iOS 4.0 and later.

See Also
`@property focusPointOfInterestSupported` (page 113)

Declared In
`AVCaptureDevice.h`

focusPointOfInterestSupported

Indicates whether the device supports a point of interest for focus. (read-only)

```
@property(nonatomic, readonly, getter=isFocusPointOfInterestSupported) BOOL
    focusPointOfInterestSupported
```

Discussion

Availability
Available in iOS 4.0 and later.

See Also
`@property focusPointOfInterest` (page 113)
`- isFocusModeSupported:` (page 119)

Declared In
`AVCaptureDevice.h`

hasFlash

Indicates whether the capture device has a flash. (read-only)

```
@property(nonatomic, readonly) BOOL hasFlash
```

Discussion

Availability
Available in iOS 4.0 and later.

See Also
`@property flashMode` (page 112)
`- isFlashModeSupported:` (page 119)

Declared In
`AVCaptureDevice.h`

hasTorch

A Boolean value that specifies whether the capture device has a torch. (read-only)

```
@property(nonatomic, readonly) BOOL hasTorch
```

Discussion

Availability
Available in iOS 4.0 and later.

See Also
`@property torchMode` (page 115)
`- isTorchModeSupported:` (page 120)

Declared In
`AVCaptureDevice.h`

localizedName

(read-only)

```
@property(nonatomic, readonly) NSString *localizedName
```

Discussion

Availability
Available in iOS 4.0 and later.

See Also
`@property modelID` (page 115)
`@property uniqueID` (page 116)

Declared In
`AVCaptureDevice.h`

modelID

(read-only)

```
@property(nonatomic, readonly) NSString *modelID
```

Discussion

Availability
Available in iOS 4.0 and later.

See Also
`@property localizedName` (page 114)
`@property uniqueID` (page 116)

Declared In
`AVCaptureDevice.h`

position

(read-only)

```
@property(nonatomic, readonly) AVCaptureDevicePosition position
```

Discussion
See "Capture Device Position" (page 122) for possible values.

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureDevice.h`

torchMode

The current torch mode.

```
@property(nonatomic) AVCaptureTorchMode torchMode
```

Discussion
See "Torch Modes" (page 124) for possible values.

Availability
Available in iOS 4.0 and later.

See Also
`@property hasTorch` (page 114)
`- isTorchModeSupported:` (page 120)
`- lockForConfiguration:` (page 121)

Declared In
`AVCaptureDevice.h`

uniqueID

(read-only)

```
@property(nonatomic, readonly) NSString *uniqueID
```

Discussion

Availability
Available in iOS 4.0 and later.

See Also
`@property localizedName` (page 114)

Declared In
`AVCaptureDevice.h`

whiteBalanceMode

The current white balance mode.

```
@property(nonatomic) AVCaptureWhiteBalanceMode whiteBalanceMode
```

Discussion
See "White Balance Modes" (page 126) for possible values.

Availability
Available in iOS 4.0 and later.

See Also
`- isWhiteBalanceModeSupported:` (page 120)
`@property adjustingWhiteBalance` (page 110)
`- lockForConfiguration:` (page 121)

Declared In
`AVCaptureDevice.h`

Class Methods

defaultDeviceWithMediaType:

Returns the default device used to capture data of a given media type.

```
+ (AVCaptureDevice *)defaultDeviceWithMediaType:(NSString *)mediaType
```

Parameters

mediaType

A media type identifier.

For possible values, see *AV Foundation Constants Reference*.

Return Value
The default device used to capture data of the type indicated by *mediaType*.

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
AVCaptureDevice.h

devices

Returns an array containing the available capture devices on the system.

```
+ (NSArray *)devices
```

Return Value
An array containing the available capture devices on the system

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
AVCaptureDevice.h

devicesWithMediaType:

Returns an array containing the devices able to capture data of a given media type

```
+ (NSArray *)devicesWithMediaType:(NSString *)mediaType
```

Parameters
mediaType
A media type identifier.
For possible values, see *AV Foundation Constants Reference*.

Return Value
An array containing the devices able to capture data of the type indicated by *mediaType*.

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
AVCaptureDevice.h

deviceWithUniqueID:

Returns the device with a given ID.

```
+ (AVCaptureDevice *)deviceWithUniqueID:(NSString *)deviceUniqueID
```

Parameters

deviceUniqueID

The ID of a capture device.

Return Value

The device with ID *deviceUniqueID*.

Discussion

Availability

Available in iOS 4.0 and later.

See Also

`@property uniqueID` (page 116)

Declared In

`AVCaptureDevice.h`

Instance Methods

hasMediaType:

```
- (BOOL)hasMediaType:(NSString *)mediaType
```

Parameters

mediaType

Return Value

Discussion

Availability

Available in iOS 4.0 and later.

Declared In

`AVCaptureDevice.h`

isExposureModeSupported:

Returns a Boolean value that indicates whether the given exposure mode is supported.

```
- (BOOL)isExposureModeSupported:(AVCaptureExposureMode)exposureMode
```

Parameters

exposureMode

An exposure mode. See "Exposure Modes" (page 125) for possible values.

Return Value

`YES` if *exposureMode* is supported, otherwise `NO`.

Discussion

Availability
Available in iOS 4.0 and later.

See Also
`@property exposureMode` (page 111)
`@property exposurePointOfInterestSupported` (page 112)

Declared In
`AVCaptureDevice.h`

isFlashModeSupported:

Returns a Boolean value that indicates whether the given flash mode is supported.

```
- (BOOL)isFlashModeSupported:(AVCaptureFlashMode)flashMode
```

Parameters
flashMode
A flash mode. See "Flash Modes" (page 123) for possible values.

Return Value
`YES` if *flashMode* is supported, otherwise `NO`.

Discussion

Availability
Available in iOS 4.0 and later.

See Also
`@property hasFlash` (page 113)
`@property flashMode` (page 112)

Declared In
`AVCaptureDevice.h`

isFocusModeSupported:

Returns a Boolean value that indicates whether the given focus mode is supported.

```
- (BOOL)isFocusModeSupported:(AVCaptureFocusMode)focusMode
```

Parameters
focusMode
A focus mode. See "Focus Modes" (page 124) for possible values.

Return Value
`YES` if *focusMode* is supported, otherwise `NO`.

Discussion

Availability
Available in iOS 4.0 and later.

See Also
`@property focusMode` (page 112)
`@property adjustingFocus` (page 110)

Declared In
`AVCaptureDevice.h`

isTorchModeSupported:

Returns a Boolean value that indicates whether the given torch mode is supported.

```
- (BOOL)isTorchModeSupported:(AVCaptureTorchMode)torchMode
```

Parameters
torchMode
A focus mode. See "Torch Modes" (page 124) for possible values.

Return Value
`YES` if *torchMode* is supported, otherwise `NO`.

Discussion

Availability
Available in iOS 4.0 and later.

See Also
`@property torchMode` (page 115)

Declared In
`AVCaptureDevice.h`

isWhiteBalanceModeSupported:

Returns a Boolean value that indicates whether the given white balance mode is supported.

```
- (BOOL)isWhiteBalanceModeSupported:(AVCaptureWhiteBalanceMode)whiteBalanceMode
```

Parameters
whiteBalanceMode
A focus mode. See "White Balance Modes" (page 126) for possible values.

Return Value
`YES` if *whiteBalanceMode* is supported, otherwise `NO`.

Discussion

Availability
Available in iOS 4.0 and later.

See Also
`@property whiteBalanceMode` (page 116)

Declared In
`AVCaptureDevice.h`

lockForConfiguration:

Attempts to acquire a lock on the capture device.

```
- (BOOL)lockForConfiguration:(NSError **)outError
```

Parameters

outError

If a lock cannot be acquired, upon return contains an `NSError` object that describes the problem.

Return Value

`YES` if a lock was acquired, otherwise `NO`.

Discussion

In order to set properties on a capture device (`focusMode` (page 112), `exposureMode` (page 111), and so on), you must first acquire a lock on the device.

Special Considerations

You should only hold the device lock if you require settable device properties to remain unchanged. Holding the device lock unnecessarily may degrade capture quality in other applications sharing the device.

Availability

Available in iOS 4.0 and later.

See Also

`- unlockForConfiguration` (page 121)

Declared In

`AVCaptureDevice.h`

supportsAVCaptureSessionPreset:

```
- (BOOL)supportsAVCaptureSessionPreset:(NSString *)preset
```

Parameters

preset

Return Value

Discussion

Availability

Available in iOS 4.0 and later.

Declared In

`AVCaptureDevice.h`

unlockForConfiguration

Relinquishes a lock on a device.

```
- (void)unlockForConfiguration
```

Discussion

Availability
Available in iOS 4.0 and later.

See Also
`- lockForConfiguration:` (page 121)

Declared In
`AVCaptureDevice.h`

Constants

AVCaptureDevicePosition

A type to specify the position of a capture device.

```
typedef NSInteger AVCaptureDevicePosition;
```

Discussion
See "Capture Device Position" (page 122) for possible values.

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureDevice.h`

Capture Device Position

Constants to specify the position of a capture device.

```
enum {
    AVCaptureDevicePositionBack  = 1,
    AVCaptureDevicePositionFront = 2
};
```

Constants

`AVCaptureDevicePositionBack`
The capture device is on the back of the unit.
Available in iOS 4.0 and later.
Declared in `AVCaptureDevice.h`.

`AVCaptureDevicePositionFront`
The capture device is on the front of the unit.
Available in iOS 4.0 and later.
Declared in `AVCaptureDevice.h`.

AVCaptureFlashMode

A type to specify the flash mode of a capture device.

```
typedef NSInteger AVCaptureFlashMode;
```

Discussion
See "Flash Modes" (page 123) for possible values.

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureDevice.h`

Flash Modes

Constants to specify the flash mode of a capture device.

```
enum {
    AVCaptureFlashModeOff    = 0,
    AVCaptureFlashModeOn     = 1,
    AVCaptureFlashModeAuto   = 2
};
```

Constants

`AVCaptureFlashModeOff`
The capture device flash is always off.
Available in iOS 4.0 and later.
Declared in `AVCaptureDevice.h`.

`AVCaptureFlashModeOn`
The capture device flash is always on.
Available in iOS 4.0 and later.
Declared in `AVCaptureDevice.h`.

`AVCaptureFlashModeAuto`
The capture device continuously monitors light levels and uses the flash when necessary.
Available in iOS 4.0 and later.
Declared in `AVCaptureDevice.h`.

AVCaptureTorchMode

A type to specify the torch mode of a capture device.

```
typedef NSInteger AVCaptureTorchMode;
```

Discussion
See "Torch Modes" (page 124) for possible values.

Availability
Available in iOS 4.0 and later.

Declared In
AVCaptureDevice.h

Torch Modes

Constants to specify the direction in which a capture device faces

```
enum {
    AVCaptureTorchModeOff    = 0,
    AVCaptureTorchModeOn     = 1,
    AVCaptureTorchModeAuto   = 2
};
```

Constants

AVCaptureTorchModeOff
The capture device torch is always off.
Available in iOS 4.0 and later.
Declared in AVCaptureDevice.h.

AVCaptureTorchModeOn
The capture device torch is always on.
Available in iOS 4.0 and later.
Declared in AVCaptureDevice.h.

AVCaptureTorchModeAuto
The capture device continuously monitors light levels and uses the torch when necessary.
Available in iOS 4.0 and later.
Declared in AVCaptureDevice.h.

AVCaptureFocusMode;

A type to specify the focus mode of a capture device.

```
typedef NSInteger AVCaptureFocusMode;
```

Discussion
See "Focus Modes" (page 124) for possible values.

Availability
Available in iOS 4.0 and later.

Declared In
AVCaptureDevice.h

Focus Modes

Constants to specify the focus mode of a capture device.

```
enum {
    AVCaptureFocusModeLocked                = 0,
    AVCaptureFocusModeAutoFocus             = 1,
    AVCaptureFocusModeContinuousAutoFocus   = 2,
};
```

Constants

`AVCaptureFocusModeLocked`

The focus is locked.

Available in iOS 4.0 and later.

Declared in `AVCaptureDevice.h`.

`AVCaptureFocusModeAutoFocus`

The capture device performs an autofocus operation now.

Available in iOS 4.0 and later.

Declared in `AVCaptureDevice.h`.

`AVCaptureFocusModeContinuousAutoFocus`

The capture device continuously monitors focus and auto focuses when necessary.

Available in iOS 4.0 and later.

Declared in `AVCaptureDevice.h`.

AVCaptureExposureMode

A type to specify the exposure mode of a capture device.

```
typedef NSInteger AVCaptureExposureMode;
```

Discussion

See "Exposure Modes" (page 125) for possible values.

Availability

Available in iOS 4.0 and later.

Declared In

`AVCaptureDevice.h`

Exposure Modes

Constants to specify the exposure mode of a capture device.

```
enum {
    AVCaptureExposureModeLocked                     = 0,
    AVCaptureExposureModeAutoExpose                 = 1,
    AVCaptureExposureModeContinuousAutoExposure     = 2,
};
```

Constants

`AVCaptureExposureModeLocked`

The exposure setting is locked.

Available in iOS 4.0 and later.

Declared in `AVCaptureDevice.h`.

`AVCaptureExposureModeAutoExpose`
The device performs an auto-expose operation now.

Available in iOS 4.0 and later.

Declared in `AVCaptureDevice.h`.

`AVCaptureExposureModeContinuousAutoExposure`
The device continuously monitors exposure levels and auto exposes when necessary.

Available in iOS 4.0 and later.

Declared in `AVCaptureDevice.h`.

AVCaptureWhiteBalanceMode

A type to specify the white balance mode of a capture device.

```
typedef NSInteger AVCaptureWhiteBalanceMode;
```

Discussion
See "White Balance Modes" (page 126) for possible values.

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureDevice.h`

White Balance Modes

Constants to specify the white balance mode of a capture device.

```
enum {
    AVCaptureWhiteBalanceModeLocked             = 0,
    AVCaptureWhiteBalanceModeAutoWhiteBalance   = 1,
    AVCaptureWhiteBalanceModeContinuousAutoWhiteBalance = 2,
};
```

Constants
`AVCaptureWhiteBalanceModeLocked`
The white balance setting is locked.

Available in iOS 4.0 and later.

Declared in `AVCaptureDevice.h`.

`AVCaptureWhiteBalanceModeAutoWhiteBalance`
The device performs an auto white balance operation now.

Available in iOS 4.0 and later.

Declared in `AVCaptureDevice.h`.

`AVCaptureWhiteBalanceModeContinuousAutoWhiteBalance`
The device continuously monitors white balance and adjusts when necessary.

Available in iOS 4.0 and later.

Declared in `AVCaptureDevice.h`.

Notifications

AVCaptureDeviceWasConnectedNotification

Notification that is posted when a new device becomes available.

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureDevice.h`

AVCaptureDeviceWasDisconnectedNotification

Notification that is posted when an existing device becomes unavailable.

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureDevice.h`

AVCaptureFileOutput Class Reference

Inherits from	AVCaptureOutput : NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/AVFoundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	AVCaptureOutput.h

Overview

`AVCaptureFileOutput` is an abstract sub-class of `AVCaptureOutput` that describes a file output destination to an `AVCaptureSession`. You use an instance of its concrete subclass, `AVCaptureMovieFileOutput`, to save capture output to a QuickTime movie file.

Tasks

Managing Recording

`- startRecordingToOutputFileURL:recordingDelegate:` (page 132)
Starts recording to a given URL.

`- stopRecording` (page 132)
Stops recording.

`recording` (page 132) *property*
Indicates whether recording is in progress.

Configuration

`maxRecordedDuration` (page 130) *property*
The longest duration allowed for the recording.

`maxRecordedFileSize` (page 130) *property*
The maximum file size allowed for the recording.

`minFreeDiskSpaceLimit` (page 131) *property*
The minimum available free disk space that must be available for recording to continue.

Information About Output

`outputFileURL` (page 131) *property*
The URL to which output is directed. (read-only)

`recordedDuration` (page 131) *property*
The total duration recorded to the current output file. (read-only)

`recordedFileSize` (page 131) *property*
The total file size recorded to the current output file. (read-only)

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C Programming Language*.

maxRecordedDuration

The longest duration allowed for the recording.

```
@property(nonatomic) CMTime maxRecordedDuration
```

Discussion
If the limit is reached, `outputFileURL` (page 131) is set to `nil`, and the `captureOutput:didFinishRecordingToOutputFileAtURL:fromConnections:error:` delegate method is invoked with an appropriate error.

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureOutput.h`

maxRecordedFileSize

The maximum file size allowed for the recording.

```
@property(nonatomic) int64_t maxRecordedFileSize
```

Discussion
If the limit is reached, `outputFileURL` (page 131) is set to `nil`, and the `captureOutput:didFinishRecordingToOutputFileAtURL:fromConnections:error:` delegate method is invoked with an appropriate error.

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureOutput.h`

minFreeDiskSpaceLimit

The minimum available free disk space that must be available for recording to continue.

```
@property(nonatomic) int64_t minFreeDiskSpaceLimit
```

Discussion
If the limit is reached, `outputFileURL` (page 131) is set to `nil`, and the `captureOutput:didFinishRecordingToOutputFileAtURL:fromConnections:error:` delegate method is invoked with an appropriate error.

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureOutput.h`

outputFileURL

The URL to which output is directed. (read-only)

```
@property(nonatomic, readonly) NSURL *outputFileURL
```

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureOutput.h`

recordedDuration

The total duration recorded to the current output file. (read-only)

```
@property(nonatomic, readonly) CMTime recordedDuration
```

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureOutput.h`

recordedFileSize

The total file size recorded to the current output file. (read-only)

```
@property(nonatomic, readonly) int64_t recordedFileSize
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
AVCaptureOutput.h

recording

Indicates whether recording is in progress.

```
@property(nonatomic, readonly, getter=isRecording) BOOL recording;
```

Availability
Available in iOS 4.0 and later.

Declared In
AVCaptureOutput.h

Instance Methods

startRecordingToOutputFileURL:recordingDelegate:

Starts recording to a given URL.

```
- (void)startRecordingToOutputFileURL:(NSURL *)outputFileURL recordingDelegate:(id
    < AVCaptureFileOutputRecordingDelegate >)delegate
```

Parameters

outputFileURL
The URL to which output is directed.

delegate
A object to serve as delegate for the recording session.

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
AVCaptureOutput.h

stopRecording

Stops recording.

```
- (void)stopRecording
```

Availability
Available in iOS 4.0 and later.

Declared In
AVCaptureOutput.h

AVCaptureInput Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/AVFoundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	AVCaptureInput.h

Overview

`AVCaptureInput` is an abstract base-class describing an input data source to an `AVCaptureSession` object.

To associate an `AVCaptureInput` object with a session, call `addInput:` (page 142) on the session.

`AVCaptureInput` objects have one or more ports (instances of `AVCaptureInputPort`), one for each data stream they can produce. For example, an `AVCaptureDevice` object presenting one video data stream has one port.

Tasks

Accessing the Ports

`ports` (page 133) *property*
The capture input's ports. (read-only)

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C Programming Language*.

ports

The capture input's ports. (read-only)

```
@property(nonatomic, readonly) NSArray *ports
```

Discussion
The array contains one or more instances of `AVCaptureInputPort`.

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureInput.h`

Notifications

AVCaptureInputPortFormatDescriptionDidChangeNotification

Posted if the format description of a capture input port changes.

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureInput.h`

AVCaptureMovieFileOutput Class Reference

Inherits from	AVCaptureFileOutput : AVCaptureOutput : NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/AVFoundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	AVCaptureOutput.h

Overview

`AVCaptureMovieFileOutput` is a concrete sub-class of `AVCaptureFileOutput` you use to capture data to a QuickTime movie.

The timeMapping.target.start of the first track segment must be `kCMTimeZero`, and the `timeMapping.target.start` of each subsequent track segment must equal `CMTimeRangeGetEnd(<#the previous AVCompositionTrackSegment's timeMapping.target#>)`. You can use `validateTrackSegments:error:` (page 204) to ensure that an array of track segments conforms to this rule.

Tasks

Movie Configuration

`movieFragmentInterval` (page 136) *property*
Indicates the number of seconds of output that are written per fragment.

`metadata` (page 136) *property*
The metadata for the output file.

Properties

For more about Objective-C properties, see “Properties” in *The Objective-C Programming Language*.

metadata

The metadata for the output file.

```
@property(nonatomic, copy) NSArray *metadata
```

Discussion
The array contains `AVMetadataItem` objects. You use this array to add metadata such as copyright, creation date, and so on, to the recorded movie file.

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureOutput.h`

movieFragmentInterval

Indicates the number of seconds of output that are written per fragment.

```
@property(nonatomic) CMTime movieFragmentInterval
```

Discussion
The default is 10 seconds. Set to `kCMTimeInvalid` to disable movie fragment writing (not typically recommended).

A QuickTime movie is comprised of media samples and a sample table identifying their location in the file. A movie file without a sample table is unreadable.

In a processed file, the sample table typically appears at the beginning of the file. It may also appear at the end of the file, in which case the header contains a pointer to the sample table at the end. When a new movie file is being recorded, it is not possible to write the sample table since the size of the file is not yet known. Instead, the table is must be written when recording is complete. If no other action is taken, this means that if the recording does not complete successfully (for example, in the event of a crash), the file data is unusable (because there is no sample table). By periodically inserting "movie fragments" into the movie file, the sample table can be built up incrementally. This means that if the file is not written completely, the movie file is still usable (up to the point where the last fragment was written).

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureOutput.h`

AVCaptureOutput Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/AVFoundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	AVCaptureOutput.h

Overview

`AVCaptureOutput` is an abstract base-class describing an output destination of an `AVCaptureSession` object.

`AVCaptureOutput` provides an abstract interface for connecting capture output destinations, such as files and video previews, to an capture session (an instance of `AVCaptureSession`). A capture output can have multiple connections represented by `AVCaptureConnection` objects, one for each stream of media that it receives from a capture input (an instance of `AVCaptureInput`). A capture output does not have any connections when it is first created. When you add an output to a capture session, connections are created that map media data from that session's inputs to its outputs.

You can add concrete `AVCaptureOutput` instances to an capture session using `addOutput:` (page 143).

Tasks

Accessing Connections

`connections` (page 138) *property*
The capture output object's connections. (read-only)

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C Programming Language*.

connections

The capture output object's connections. (read-only)

```
@property(nonatomic, readonly) NSArray *connections
```

Discussion
The value of this property is an array of `AVCaptureConnection` objects, each describing the mapping between the receiver and the capture input ports (see `AVCaptureInputPort`) of one or more capture inputs (see `AVCaptureInput`).

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureOutput.h`

AVCaptureSession Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/AVFoundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	AVCaptureSession.h

Overview

You use an `AVCaptureSession` object to coordinate the flow of data from AV input devices to outputs.

To perform a real-time or offline capture, you instantiate an `AVCaptureSession` object and add appropriate inputs (such as `AVCaptureDeviceInput`), and outputs (such as `AVCaptureMovieFileOutput`). The following code fragment illustrates how to configure a capture device to record audio:

```
AVCaptureSession *captureSession = [[AVCaptureSession alloc] init];
AVCaptureDevice *audioCaptureDevice = [AVCaptureDevice
defaultDeviceWithMediaType:AVMediaTypeAudio];
NSError *error = nil;
AVCaptureDeviceInput *audioInput = [AVCaptureDeviceInput
deviceInputWithDevice:audioCaptureDevice error:&error];
if (audioInput) {
    [captureSession addInput:audioInput];
}
else {
    // Handle the failure.
}
```

You invoke `startRunning` (page 146) to start the flow of data from the inputs to the outputs, and `stopRunning` (page 146) to stop the flow. You use the `sessionPreset` (page 142) property to customize the quality of the output.

Tasks

Managing Inputs and Outputs

`inputs` (page 141) *property*
The capture session's inputs. (read-only)

`outputs` (page 141) *property*
The capture session's outputs. (read-only)

- `addInput:` (page 142)
Adds a given input to the session.

- `addOutput:` (page 143)
Adds a given output to the session.

- `canAddInput:` (page 144)
Returns a Boolean value that indicates whether a given input can be added to the session.

- `canAddOutput:` (page 144)
Returns a Boolean value that indicates whether a given output can be added to the session.

- `removeInput:` (page 145)
Removes a given input.

- `removeOutput:` (page 146)
Removes a given output.

Managing Running State

- `startRunning` (page 146)
Tells the receiver to start running.

- `stopRunning` (page 146)
Tells the receiver to stop running.

`running` (page 142) *property*
Indicates whether the receiver is running. (read-only)

`interrupted` (page 141) *property*
Indicates whether the receiver has been interrupted. (read-only)

Configuration Change

- `beginConfiguration` (page 143)
Indicates the start of a set of configuration changes to be made atomically.

- `commitConfiguration` (page 145)
Commits a set of configuration changes.

Managing Session Presets

`sessionPreset` (page 142) *property*
The capture session's preset.

- `canSetSessionPreset:` (page 145)
Returns a Boolean value that indicates whether the receiver can use the given preset.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C Programming Language*.

inputs

The capture session's inputs. (read-only)

```
@property(nonatomic, readonly) NSArray *inputs
```

Discussion
The array contains instances of subclasses of `AVCaptureInput`.

Availability
Available in iOS 4.0 and later.

See Also
- `addInput:` (page 142)
- `canAddInput:` (page 144)
- `removeInput:` (page 145)

Declared In
`AVCaptureSession.h`

interrupted

Indicates whether the receiver has been interrupted. (read-only)

```
@property(nonatomic, readonly, getter=isInterrupted) BOOL interrupted
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureSession.h`

outputs

The capture session's outputs. (read-only)

```
@property(nonatomic, readonly) NSArray *outputs
```

Discussion
The array contains instances of subclasses of `AVCaptureOutput`.

Availability
Available in iOS 4.0 and later.

See Also
- `addOutput:` (page 143)
- `canAddOutput:` (page 144)
- `removeOutput:` (page 146)

Declared In
`AVCaptureSession.h`

running

Indicates whether the receiver is running. (read-only)

```
@property(nonatomic, readonly, getter=isRunning) BOOL running
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureSession.h`

sessionPreset

The capture session's preset.

```
@property(nonatomic, copy) NSString *sessionPreset
```

Discussion
For possible values of *`sessionPreset`*, see "Video Input Presets" (page 148).

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureSession.h`

Instance Methods

addInput:

Adds a given input to the session.

```
- (void)addInput:(AVCaptureInput *)input
```

Parameters
`input`
An input to add to the session.

Discussion

Availability
Available in iOS 4.0 and later.

See Also
- `canAddInput:` (page 144)
- `addOutput:` (page 143)
- `removeInput:` (page 145)

Declared In
`AVCaptureSession.h`

addOutput:

Adds a given output to the session.

```
- (void)addOutput:(AVCaptureOutput *)output
```

Parameters

output
An output to add to the session.

Discussion

Availability
Available in iOS 4.0 and later.

See Also
- `canAddOutput:` (page 144)
- `addInput:` (page 142)
- `removeOutput:` (page 146)

Declared In
`AVCaptureSession.h`

beginConfiguration

Indicates the start of a set of configuration changes to be made atomically.

```
- (void)beginConfiguration
```

Discussion
You use `beginConfiguration` and `commitConfiguration` (page 145) to batch multiple configuration operations on a running session into an atomic update.

After calling `beginConfiguration`, you can for example add or remove outputs, alter the `sessionPreset` (page 142), or configure individual capture input or output properties. No changes are actually made until you invoke `commitConfiguration` (page 145), at which time they are applied together.

Availability
Available in iOS 4.0 and later.

See Also
- commitConfiguration (page 145)

Declared In
AVCaptureSession.h

canAddInput:

Returns a Boolean value that indicates whether a given input can be added to the session.

```
- (BOOL)canAddInput:(AVCaptureInput *)input
```

Parameters

input

An input that you want to add to the session.

Return Value
YES if *input* can be added to the session, otherwise NO.

Discussion

Availability
Available in iOS 4.0 and later.

See Also
- addInput: (page 142)

Declared In
AVCaptureSession.h

canAddOutput:

Returns a Boolean value that indicates whether a given output can be added to the session.

```
- (BOOL)canAddOutput:(AVCaptureOutput *)output
```

Parameters

output

An output that you want to add to the session.

Return Value
YES if *output* can be added to the session, otherwise NO.

Discussion

Availability
Available in iOS 4.0 and later.

See Also
- addOutput: (page 143)

Declared In
AVCaptureSession.h

canSetSessionPreset:

Returns a Boolean value that indicates whether the receiver can use the given preset.

```
- (BOOL)canSetSessionPreset:(NSString *)preset
```

Parameters

preset

A preset you would like to set for the receiver. For possible values, see "Video Input Presets" (page 148).

Return Value

`YES` if the receiver can use *preset*, otherwise `NO`.

Discussion

Availability

Available in iOS 4.0 and later.

Declared In

`AVCaptureSession.h`

commitConfiguration

Commits a set of configuration changes.

```
- (void)commitConfiguration
```

Discussion

For discussion, see `beginConfiguration` (page 143).

Availability

Available in iOS 4.0 and later.

Declared In

`AVCaptureSession.h`

removeInput:

Removes a given input.

```
- (void)removeInput:(AVCaptureInput *)input
```

Parameters

input

An input to remove from the receiver.

Discussion

Availability

Available in iOS 4.0 and later.

See Also

`- addInput:` (page 142)

Declared In
AVCaptureSession.h

removeOutput:

Removes a given output.

```
- (void)removeOutput:(AVCaptureOutput *)output
```

Parameters

output
An output to remove from the receiver.

Discussion

Availability
Available in iOS 4.0 and later.

See Also
- addOutput: (page 143)

Declared In
AVCaptureSession.h

startRunning

Tells the receiver to start running.

```
- (void)startRunning
```

Discussion
startRunning and stopRunning (page 146) are asynchronous operations. If an error occurs occur during a capture session, you receive an AVCaptureSessionRuntimeErrorNotification (page 149).

Availability
Available in iOS 4.0 and later.

See Also
- stopRunning (page 146)

Declared In
AVCaptureSession.h

stopRunning

Tells the receiver to stop running.

```
- (void)stopRunning
```

Discussion
startRunning (page 146) and stopRunning are asynchronous operations. If an error occurs occur during a capture session, you receive an AVCaptureSessionRuntimeErrorNotification (page 149).

Availability
Available in iOS 4.0 and later.

See Also
- `startRunning` (page 146)

Declared In
`AVCaptureSession.h`

Constants

AVCaptureVideoOrientation

Constants to specify the device orientation during video capture.

```
enum {
    AVCaptureVideoOrientationPortrait               = 1,
    AVCaptureVideoOrientationPortraitUpsideDown,
    AVCaptureVideoOrientationLandscapeLeft,
    AVCaptureVideoOrientationLandscapeRight,
};
typedef NSInteger AVCaptureVideoOrientation;
```

Constants

`AVCaptureVideoOrientationPortrait`
Indicates that the video input is oriented vertically, with the device's home button on the bottom.
Available in iOS 4.0 and later.
Declared in `AVCaptureSession.h`.

`AVCaptureVideoOrientationPortraitUpsideDown`
Indicates that the video input is oriented vertically, with the device's home button on the top.
Available in iOS 4.0 and later.
Declared in `AVCaptureSession.h`.

`AVCaptureVideoOrientationLandscapeLeft`
Indicates that the video input is oriented vertically, *with the device's home button on the right*.
Available in iOS 4.0 and later.
Declared in `AVCaptureSession.h`.

`AVCaptureVideoOrientationLandscapeRight`
Indicates that the video input is oriented vertically, *with the device's home button on the left*.
Available in iOS 4.0 and later.
Declared in `AVCaptureSession.h`.

Notification User Info Key

Key to retrieve information from a notification from a capture session.

```
NSString *const AVCaptureSessionErrorKey;
```

Constants

`AVCaptureSessionErrorKey`

Key to retrieve the error object from the user info dictionary of an `AVCaptureSessionRuntimeErrorNotification` (page 149).

Available in iOS 4.0 and later.

Declared in `AVCaptureSession.h`.

Video Input Presets

Constants to define capture setting presets.

```
NSString *const AVCaptureSessionPresetPhoto;
NSString *const AVCaptureSessionPresetHigh;
NSString *const AVCaptureSessionPresetMedium;
NSString *const AVCaptureSessionPresetLow;
NSString *const AVCaptureSessionPreset640x480;
NSString *const AVCaptureSessionPreset1280x720;
```

Constants

`AVCaptureSessionPresetPhoto`

Available in iOS 4.0 and later.

Declared in `AVCaptureSession.h`.

`AVCaptureSessionPresetHigh`

Available in iOS 4.0 and later.

Declared in `AVCaptureSession.h`.

`AVCaptureSessionPresetMedium`

Available in iOS 4.0 and later.

Declared in `AVCaptureSession.h`.

`AVCaptureSessionPresetLow`

Available in iOS 4.0 and later.

Declared in `AVCaptureSession.h`.

`AVCaptureSessionPreset640x480`

Available in iOS 4.0 and later.

Declared in `AVCaptureSession.h`.

`AVCaptureSessionPreset1280x720`

Available in iOS 4.0 and later.

Declared in `AVCaptureSession.h`.

Notifications

AVCaptureSessionRuntimeErrorNotification

Posted if an error occurred during a capture session.

You retrieve the underlying error from the notification's user info dictionary using the key `AVCaptureSessionErrorKey` (page 148).

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureSession.h`

AVCaptureSessionDidStartRunningNotification

Posted when a capture session starts.

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureSession.h`

AVCaptureSessionDidStopRunningNotification

Posted when a capture session stops.

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureSession.h`

AVCaptureSessionWasInterruptedNotification

Posted if a capture session is interrupted.

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureSession.h`

AVCaptureSessionInterruptionEndedNotification

Posted if an interruption to a capture session finishes.

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureSession.h`

AVCaptureStillImageOutput Class Reference

Inherits from	AVCaptureOutput : NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/AVFoundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	AVCaptureOutput.h

Overview

`AVCaptureStillImageOutput` is a concrete sub-class of `AVCaptureOutput` that you use to capture a high-quality still image with accompanying metadata.

Tasks

Capturing an Image

`- captureStillImageAsynchronouslyFromConnection:completionHandler:` (page 154)
Initiates a still image capture and returns immediately.

Image Configuration

`outputSettings` (page 152) *property*
The compression settings for the output.

`availableImageDataCVPixelFormatTypes` (page 152) *property*
The supported image pixel formats that can be specified in `outputSettings` (page 152). (read-only)

`availableImageDataCodecTypes` (page 152) *property*
The supported image codec formats that can be specified in `outputSettings` (page 152). (read-only)

Image Format Conversion

`+ jpegStillImageNSDataRepresentation:` (page 153)
Returns an `NSData` representation of a still image data and metadata attachments in a JPEG sample buffer.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C Programming Language*.

availableImageDataCodecTypes

The supported image codec formats that can be specified in `outputSettings` (page 152). (read-only)

```
@property(nonatomic, readonly) NSArray *availableImageDataCodecTypes
```

Discussion
The value of this property is an array of `NSString` objects that you can use as values for the `AVVideoCodecKey` (page 316) in the `outputSettings` (page 152) property.

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureOutput.h`

availableImageDataCVPixelFormatTypes

The supported image pixel formats that can be specified in `outputSettings` (page 152). (read-only)

```
@property(nonatomic, readonly) NSArray *availableImageDataCVPixelFormatTypes
```

Discussion
The value of this property is an array of `NSNumber` objects that you can use as values for the `kCVPixelBufferPixelFormatTypeKey` in the `outputSettings` (page 152) property.

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureOutput.h`

outputSettings

The compression settings for the output.

```
@property(nonatomic, copy) NSDictionary *outputSettings
```

Discussion
You specify the compression settings using keys from AVVideoSettings.h, or a dictionary of pixel buffer attributes using keys from CVPixelBuffer.h.

Currently the only supported keys are `AVVideoCodecKey` (page 316) and `kCVPixelBufferPixelFormatTypeKey`. The recommended values are `kCMVideoCodecType_JPEG`, `kCVPixelFormatType_420YpCbCr8BiPlanarFullRange` and `kCVPixelFormatType_32BGRA`.

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureOutput.h`

Class Methods

jpegStillImageNSDataRepresentation:

Returns an `NSData` representation of a still image data and metadata attachments in a JPEG sample buffer.

```
+ (NSData *)jpegStillImageNSDataRepresentation:(CMSampleBufferRef)jpegSampleBuffer
```

Parameters

jpegSampleBuffer

The sample buffer carrying JPEG image data, optionally with Exif metadata sample buffer attachments.

This method throws an `NSInvalidArgumentException` if *jpegSampleBuffer* is `NULL` or not in the JPEG format.

Return Value
An `NSData` representation of *jpegSampleBuffer*.

Discussion
This method merges the image data and Exif metadata sample buffer attachments without re-compressing the image.

The returned `NSData` object is suitable for writing to disk.

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureOutput.h`

Instance Methods

captureStillImageAsynchronouslyFromConnection:completionHandler:

Initiates a still image capture and returns immediately.

```
- (void)captureStillImageAsynchronouslyFromConnection:(AVCaptureConnection
    *)connection completionHandler:(void (^)(CMSampleBufferRef imageDataSampleBuffer,
     NSError *error))handler
```

Parameters

connection

The connection from which to capture the image.

handler

A block to invoke after the image has been captured. The block parameters are as follows:

imageDataSampleBuffer

The data that was captured.

The buffer attachments may contain metadata appropriate to the image data format. For example, a buffer containing JPEG data may carry a `kCGImagePropertyExifDictionary` as an attachment. See ImageIO/CGImageProperties.h for a list of keys and value types.

error

If the request could not be completed, an `NSError` object that describes the problem; otherwise `nil`.

Discussion

Availability

Available in iOS 4.0 and later.

Declared In

`AVCaptureOutput.h`

AVCaptureVideoDataOutput Class Reference

Inherits from	AVCaptureOutput : NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/AVFoundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	AVCaptureOutput.h

Overview

`AVCaptureVideoDataOutput` is a concrete sub-class of `AVCaptureOutput` you use to process uncompressed frames from the video being captured, or to access compressed frames.

An instance of `AVCaptureVideoDataOutput` produces video frames you can process using other media APIs. You can access the frames with the `captureOutput:didOutputSampleBuffer:fromConnection:` delegate method.

Tasks

Configuration

`videoSettings` (page 157) *property*
The compression settings for the output.

`minFrameDuration` (page 156) *property*
The minimum frame duration.

`alwaysDiscardsLateVideoFrames` (page 156) *property*
Indicates whether video frames are dropped if they arrive late.

Managing the Delegate

`- setSampleBufferDelegate:queue:` (page 158)
Sets the sample buffer delegate and the queue on which callbacks should be invoked.

`sampleBufferDelegate` (page 157) *property*
The capture object's delegate.

`sampleBufferCallbackQueue` (page 157) *property*
The queue on which delegate callbacks should be invoked (read-only)

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C Programming Language*.

alwaysDiscardsLateVideoFrames

Indicates whether video frames are dropped if they arrive late.

```
@property(nonatomic) BOOL alwaysDiscardsLateVideoFrames
```

Discussion
When the value of this property is `YES`, the object immediately discards frames that are captured while the dispatch queue handling existing frames is blocked in the `captureOutput:didOutputSampleBuffer:fromConnection:` delegate method.

When the value of this property is `YES`, delegates are allowed more time to process old frames before new frames are discarded, but application memory usage may increase significantly as a result.

The default is `YES`.

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureOutput.h`

minFrameDuration

The minimum frame duration.

```
@property(nonatomic) CMTime minFrameDuration
```

Discussion
This property specifies the minimum duration of each video frame output by the receiver, placing a lower bound on the amount of time that should separate consecutive frames. This is equivalent to the inverse of the maximum frame rate. A value of `kCMTimeZero` or `kCMTimeInvalid` indicates an unlimited maximum frame rate.

The default value is `kCMTimeInvalid`.

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureOutput.h`

sampleBufferCallbackQueue

The queue on which delegate callbacks should be invoked (read-only)

```
@property(nonatomic, readonly) dispatch_queue_t sampleBufferCallbackQueue
```

Discussion
You set the queue using `setSampleBufferDelegate:queue:` (page 158).

Availability
Available in iOS 4.0 and later.

See Also
- `setSampleBufferDelegate:queue:` (page 158)
- `@property sampleBufferDelegate` (page 157)

Declared In
`AVCaptureOutput.h`

sampleBufferDelegate

The capture object's delegate.

```
@property(nonatomic, readonly) id<AVCaptureVideoDataOutputSampleBufferDelegate>
    sampleBufferDelegate
```

Discussion
The delegate receives sample buffers after they are captured.

You set the delegate using `setSampleBufferDelegate:queue:` (page 158).

Availability
Available in iOS 4.0 and later.

See Also
- `setSampleBufferDelegate:queue:` (page 158)
- `@property sampleBufferCallbackQueue` (page 157)

Declared In
`AVCaptureOutput.h`

videoSettings

The compression settings for the output.

```
@property(nonatomic, copy) NSDictionary *videoSettings
```

Discussion
The dictionary contains values for compression settings keys defined in `AVVideoSettings.h`, or pixel buffer attributes keys defined in <CoreVideo/CVPixelBuffer.h> (see `CVPixelBufferRef`).

If you set this property to `nil`, the video data output vends samples in the device native format.

Currently, the only supported key is `kCVPixelBufferPixelFormatTypeKey`. Recommended pixel format choices are `kCVPixelFormatType_420YpCbCr8BiPlanarVideoRange` or `kCVPixelFormatType_32BGRA`. On iPhone 3G, the recommended pixel format choices are `kCVPixelFormatType_422YpCbCr8` or `kCVPixelFormatType_32BGRA`.

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureOutput.h`

Instance Methods

setSampleBufferDelegate:queue:

Sets the sample buffer delegate and the queue on which callbacks should be invoked.

```
- (void)setSampleBufferDelegate:(id < AVCaptureVideoDataOutputSampleBufferDelegate
    >)sampleBufferDelegate queue:(dispatch_queue_t)sampleBufferCallbackQueue
```

Parameters

`sampleBufferDelegate`

The sample buffer delegate.

`sampleBufferCallbackQueue`

The queue on which callbacks should be invoked.

You must use a serial dispatch queue, to guarantee that video frames will be delivered in order. This must not be `NULL`.

Discussion
When a new video sample buffer is captured, it is sent to the sample buffer delegate using `captureOutput:didOutputSampleBuffer:fromConnection:`. All delegate methods are invokes on the specified dispatch queue. If the queue is blocked when new frames are captured, those frames will be automatically dropped at a time determined by the value of the `alwaysDiscardsLateVideoFrames` (page 156) property. This allows you to process existing frames on the same queue without having to manage the potential memory usage increases that would otherwise occur when that processing is unable to keep up with the rate of incoming frames.

If your frame processing is consistently unable to keep up with the rate of incoming frames, you should consider using the `minFrameDuration` (page 156) property, which will generally yield better performance characteristics and more consistent frame rates than frame dropping alone.

If you need to minimize the chances of frames being dropped, you should specify a queue on which a sufficiently small amount of processing is being done outside of receiving sample buffers. However, if you migrate extra processing to another queue, you are responsible for ensuring that memory usage does not grow without bound from frames that have not been processed.

Special Considerations

This method uses `dispatch_retain` and `dispatch_release` to manage the queue.

Availability
Available in iOS 4.0 and later.

See Also

`@property sampleBufferDelegate` (page 157)

`@property sampleBufferCallbackQueue` (page 157)

Declared In

`AVCaptureOutput.h`

AVCaptureVideoPreviewLayer Class Reference

Inherits from	CALayer : NSObject
Conforms to	NSCoding (CALayer) CAMediaTiming (CALayer) NSObject (NSObject)
Framework	/System/Library/Frameworks/AVFoundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	AVCaptureVideoPreviewLayer.h

Overview

`AVCaptureVideoPreviewLayer` is a subclass of `CALayer` that allows you use to display video as it is being captured by an input device.

You use this preview layer in conjunction with an AV capture session, as illustrated in the following code fragment:

```
AVCaptureSession *captureSession = <#Get a capture session#>;
AVCaptureVideoPreviewLayer *previewLayer = [AVCaptureVideoPreviewLayer
layerWithSession:captureSession];
UIView *aView = <#The view in which to present the layer#>;
previewLayer.frame = aView.bounds; // Assume you want the preview layer to fill
 the view.
[aView.layer addSublayer:previewLayer];
```

Tasks

Creating a Session

`- initWithSession:` (page 165)
Initializes a preview layer with a given capture session.

`+ layerWithSession:` (page 164)
Returns a preview layer initialized with a given capture session.

Layer Configuration

`orientation` (page 163) *property*
The layer's orientation.

`orientationSupported` (page 163) *property*
Indicates whether the layer display supports changing the orientation. (read-only)

`mirrored` (page 162) *property*
Indicates whether the layer display is mirrored.

`mirroringSupported` (page 163) *property*
Indicates whether the layer display supports mirroring. (read-only)

`automaticallyAdjustsMirroring` (page 162) *property*
Indicates whether the layer display automatically adjusts mirroring.

`videoGravity` (page 164) *property*
Indicates how the video is displayed within a player layer's bounds rect.

`session` (page 164) *property*
The capture session with which the layer is associated.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C Programming Language*.

automaticallyAdjustsMirroring

Indicates whether the layer display automatically adjusts mirroring.

```
@property(nonatomic) BOOL automaticallyAdjustsMirroring
```

Discussion
The default value is `YES`.

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureVideoPreviewLayer.h`

mirrored

Indicates whether the layer display is mirrored.

```
@property(nonatomic, getter=isMirrored) BOOL mirrored
```

Discussion
To change the value of this property, the value of `automaticallyAdjustsMirroring` (page 162) must be `NO`.

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureVideoPreviewLayer.h`

mirroringSupported

Indicates whether the layer display supports mirroring. (read-only)

```
@property(nonatomic, readonly, getter=isMirroringSupported) BOOL mirroringSupported
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureVideoPreviewLayer.h`

orientation

The layer's orientation.

```
@property(nonatomic) AVCaptureVideoOrientation orientation
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureVideoPreviewLayer.h`

orientationSupported

Indicates whether the layer display supports changing the orientation. (read-only)

```
@property(nonatomic, readonly, getter=isOrientationSupported) BOOL
    orientationSupported
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureVideoPreviewLayer.h`

session

The capture session with which the layer is associated.

```
@property(nonatomic, retain) AVCaptureSession *session
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureVideoPreviewLayer.h`

videoGravity

Indicates how the video is displayed within a player layer's bounds rect.

```
@property(copy) NSString *videoGravity
```

Discussion
Options are `AVLayerVideoGravityResizeAspect`, `AVLayerVideoGravityResizeAspectFill` and `AVLayerVideoGravityResize`. The default is `AVLayerVideoGravityResizeAspect`.

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureVideoPreviewLayer.h`

Class Methods

layerWithSession:

Returns a preview layer initialized with a given capture session.

```
+ (id)layerWithSession:(AVCaptureSession *)session
```

Parameters

session
The capture session from which to derive the preview.

Return Value
A preview layer initialized to use *session*.

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureVideoPreviewLayer.h`

Instance Methods

initWithSession:

Initializes a preview layer with a given capture session.

```
- (id)initWithSession:(AVCaptureSession *)session
```

Parameters

session

The capture session from which to derive the preview.

Return Value

A preview layer initialized to use *session*.

Discussion

Availability

Available in iOS 4.0 and later.

Declared In

AVCaptureVideoPreviewLayer.h

AVComposition Class Reference

Inherits from	AVAsset : NSObject
Conforms to	NSMutableCopying NSCopying (AVAsset) AVAsynchronousKeyValueLoading (AVAsset) NSObject (NSObject)
Framework	/System/Library/Frameworks/AVFoundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	AVComposition.h

Overview

An `AVComposition` object combines media data from multiple file-based sources in a custom temporal arrangement, in order to present or process media data from multiple sources together. All file-based audiovisual assets are eligible to be combined, regardless of container type. The tracks in an `AVComposition` object are fixed; to change the tracks, you use an instance of its subclass, `AVMutableComposition`.

At its top-level, `AVComposition` is a collection of tracks, each presenting media of a specific media type, e.g. audio or video, according to a timeline. Each track is represented by an instance of `AVCompositionTrack`. Each track is comprised of an array of track segments, represented by instances of `AVCompositionTrackSegment`. Each segment presents a portion of the media data stored in a source container, specified by URL, a track identifier, and a time mapping. The URL specifies the source container, and the track identifier indicates the track of the source container to be presented.

The time mapping specifies the temporal range of the source track that's to be presented and also specifies the temporal range of its presentation in the composition track. If the durations of the source and destination ranges of the time mapping are the same, the media data for the segment will be presented at its natural rate. Otherwise, the segment will be presented at a rate equal to the ratio `source.duration / target.duration`.

You can access the track segments of a track using the `segments` property (an array of `AVCompositionTrackSegment` objects) of `AVCompositionTrack`. The collection of tracks with media type information for each, and each with its array of track segments (URL, track identifier, and time mapping), form a complete low-level representation of a composition. This representation can be written out by clients in any convenient form, and subsequently the composition can be reconstituted by instantiating a new `AVMutableComposition` with `AVMutableCompositionTrack` objects of the appropriate media type, each with its `segments` property set according to the stored array of URL, track identifier, and time mapping.

A higher-level interface for constructing compositions is also presented by `AVMutableComposition` and `AVMutableCompositionTrack`, offering insertion, removal, and scaling operations without direct manipulation of the trackSegment arrays of composition tracks. This interface makes use of higher-level constructs such as `AVAsset` and `AVAssetTrack`, allowing the client to make use of the same references to candidate sources that it would have created in order to inspect or preview them prior to inclusion in a composition.

Tasks

Accessing Tracks

`tracks` (page 168) *property*
An array of `AVCompositionTrack` objects contained by the composition. (read-only)

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C Programming Language*.

tracks

An array of `AVCompositionTrack` objects contained by the composition. (read-only)

```
@property(nonatomic, readonly) NSArray *tracks
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVComposition.h`

AVCompositionTrack Class Reference

Inherits from	AVAssetTrack : NSObject
Conforms to	NSCopying (AVAssetTrack) AVAsynchronousKeyValueLoading (AVAssetTrack) NSObject (NSObject)
Framework	/System/Library/Frameworks/AVFoundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	AVCompositionTrack.h

Overview

An `AVCompositionTrack` object provides the low-level representation of tracks a track in an `AVComposition` object, comprising a media type, a track identifier, and an array of `AVCompositionTrackSegment` objects, each comprising a URL, and track identifier, and a time mapping.

The `timeMapping.target.start` of the first track segment in a composition track is `kCMTimeZero`, and the `timeMapping.target.start` of each subsequent track segment equals `CMTimeRangeGetEnd(<#previousTrackSegment#>.timeMapping.target)`.

The AVFoundation framework also provides a mutable subclass, `AVMutableCompositionTrack`.

Tasks

Accessing Track Segments

`segments` (page 170) *property*
The composition track's track segments. (read-only)

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C Programming Language*.

segments

The composition track's track segments. (read-only)

```
@property(nonatomic, readonly, copy) NSArray *segments
```

Availability
Available in iOS 4.0 and later.

Declared In
`AVCompositionTrack.h`

AVCompositionTrackSegment Class Reference

Inherits from	AVAssetTrackSegment : NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/AVFoundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	AVCompositionTrackSegment.h

Overview

An `AVCompositionTrackSegment` object represents a segment of an `AVCompositionTrack` object, comprising a URL, and track identifier, and a time mapping from the source track to the composition track.

You typically use this class to save a low-level representation of a composition to a storage format of your choosing, and to reconstitute a composition from storage.

Tasks

Creating a Segment

`+ compositionTrackSegmentWithTimeRange:` (page 172)
Returns a composition track segment that presents an empty track segment.

`- initWithTimeRange:` (page 174)
Initializes a track segment that presents an empty track segment.

`+ compositionTrackSegmentWithURL:trackID:sourceTimeRange:targetTimeRange:` (page 173)
Returns a composition track segment that presents a portion of a file referenced by a given URL.

`- initWithURL:trackID:sourceTimeRange:targetTimeRange:` (page 174)
Initializes a track segment that presents a portion of a file referenced by a given URL.

Segment Properties

`sourceURL` (page 172) *property*
The container file of the media presented by the track segment. (read-only)

`sourceTrackID` (page 172) *property*
The track ID of the container file of the media presented by the track segment. (read-only)

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C Programming Language*.

sourceTrackID

The track ID of the container file of the media presented by the track segment. (read-only)

```
@property(nonatomic, readonly) CMPersistentTrackID sourceTrackID
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVCompositionTrackSegment.h`

sourceURL

The container file of the media presented by the track segment. (read-only)

```
@property(nonatomic, readonly) NSURL *sourceURL
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVCompositionTrackSegment.h`

Class Methods

compositionTrackSegmentWithTimeRange:

Returns a composition track segment that presents an empty track segment.

```
+ (AVCompositionTrackSegment
    *)compositionTrackSegmentWithTimeRange:(CMTimeRange)timeRange
```

Parameters

timeRange
The time range of the empty composition track segment.

Return Value
An composition track segment that presents an empty track segment.

Discussion
This method invokes `initWithURL:trackID:sourceTimeRange:targetTimeRange:` (page 174) with a `nil` URL, a trackID of `kCMPersistentTrackID_Invalid`, a time mapping with `source.start` and `source.duration` equal to `kCMTimeInvalid`, and with a target equal to *`timeRange`*.

This is the standard low-level representation of an empty track segment.

Availability
Available in iOS 4.0 and later.

Declared In
`AVCompositionTrackSegment.h`

compositionTrackSegmentWithURL:trackID:sourceTimeRange:targetTimeRange:

Returns a composition track segment that presents a portion of a file referenced by a given URL.

```
+ (AVCompositionTrackSegment *)compositionTrackSegmentWithURL:(NSURL *)URL
    trackID:(CMPersistentTrackID)trackID sourceTimeRange:(CMTimeRange)sourceTimeRange
     targetTimeRange:(CMTimeRange)targetTimeRange
```

Parameters

`URL`
An URL that references the container file to be presented by the track segment.

`trackID`
The track identifier that specifies the track of the container file to be presented by the track segment.

`sourceTimeRange`
The time range of the track of the container file to be presented by the track segment..

`targetTimeRange`
The time range of the composition track during which the track segment is to be presented.

Return Value
A track segment that presents a portion of a file referenced by *`URL`*.

Discussion
To specify that the segment be played at the asset's normal rate, set `source.duration == target.duration` in the time mapping. Otherwise, the segment will be played at a rate equal to the ratio `source.duration / target.duration`.

Availability
Available in iOS 4.0 and later.

Declared In
`AVCompositionTrackSegment.h`

Instance Methods

initWithTimeRange:

Initializes a track segment that presents an empty track segment.

```
- (id)initWithTimeRange:(CMTimeRange)timeRange
```

Parameters

timeRange

The time range of the empty track segment.

Return Value

A track segment that presents an empty track segment.

Discussion

This method invokes `initWithURL:trackID:sourceTimeRange:targetTimeRange:` (page 174) with a `nil` URL, a trackID of `kCMPersistentTrackID_Invalid`, a time mapping with `source.start` and `source.duration` equal to `kCMTimeInvalid`, and with a target equal to *timeRange*.

This is the standard low-level representation of an empty track segment.

Availability

Available in iOS 4.0 and later.

Declared In

`AVCompositionTrackSegment.h`

initWithURL:trackID:sourceTimeRange:targetTimeRange:

Initializes a track segment that presents a portion of a file referenced by a given URL.

```
- (id)initWithURL:(NSURL *)URL trackID:(CMPersistentTrackID)trackID
    sourceTimeRange:(CMTimeRange)sourceTimeRange
    targetTimeRange:(CMTimeRange)targetTimeRange
```

Parameters

URL

An URL that references the container file to be presented by the track segment.

trackID

The track identifier that specifies the track of the container file to be presented by the track segment.

sourceTimeRange

The time range of the track of the container file to be presented by the track segment..

targetTimeRange

The time range of the composition track during which the track segment is to be presented.

Return Value

A track segment that presents a portion of a file referenced by *URL*.

Discussion
To specify that the segment be played at the asset's normal rate, set `source.duration == target.duration` in the time mapping. Otherwise, the segment will be played at a rate equal to the ratio `source.duration / target.duration`.

Availability
Available in iOS 4.0 and later.

Declared In
`AVCompositionTrackSegment.h`

AVMetadataItem Class Reference

Inherits from	NSObject
Conforms to	NSCopying NSMutableCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/AVFoundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	AVMetadataItem.h

Overview

An `AVMetadataItem` object represents an item of metadata associated with an audiovisual asset or with one of its tracks.

Metadata items have keys that accord with the specification of the container format from which they're drawn. Full details of the metadata formats, metadata keys, and metadata key spaces supported by AV Foundation are available among the defines in AVMetadataFormat.h.

`AVAsset` and other classes provide their metadata "lazily" (see `AVAsynchronousKeyValueLoading`), meaning that you can obtain objects from those arrays without incurring overhead for items you don't ultimately inspect.

You can filter arrays of metadata items by locale or by key and key space using `metadataItemsFromArray:withLocale:` (page 182) and `metadataItemsFromArray:withKey:keySpace:` (page 182) respectively.

Tasks

Filtering Metadata Arrays

`+ metadataItemsFromArray:withKey:keySpace:` (page 182)
Returns from a given array an array of metadata items that match a specified key or key space.

`+ metadataItemsFromArray:withLocale:` (page 182)
Returns from a given array an array of metadata items that match a specified locale.

Keys and Key Spaces

`key` (page 180) *property*
Indicates the metadata item's key. (read-only)

`keySpace` (page 180) *property*
Indicates the key space of metadata item's key. (read-only)

`commonKey` (page 178) *property*
The common key of the metadata item. (read-only)

Accessing Values

`extraAttributes` (page 179) *property*
Provides a dictionary of the additional attributes. (read-only)

`locale` (page 180) *property*
Indicates the locale of the metadata item. (read-only)

`time` (page 181) *property*
Indicates the timestamp of the metadata item. (read-only)

`value` (page 181) *property*
Provides the value of the metadata item. (read-only)

`dataValue` (page 179) *property*
Provides the raw bytes of the value of the metadata item. (read-only)

Type Coercion

`dateValue` (page 179) *property*
Provides the value of the metadata item as a date. (read-only)

`numberValue` (page 180) *property*
Provides the value of the metadata item as a number. (read-only)

`stringValue` (page 181) *property*
Provides the value of the metadata item as a string. (read-only)

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C Programming Language*.

commonKey

The common key of the metadata item. (read-only)

```
@property(readonly, copy) NSString *commonKey
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVMetadataItem.h`

dataValue

Provides the raw bytes of the value of the metadata item. (read-only)

```
@property(readonly) NSData *dataValue
```

Availability
Available in iOS 4.0 and later.

Declared In
`AVMetadataItem.h`

dateValue

Provides the value of the metadata item as a date. (read-only)

```
@property(readonly) NSDate *dateValue
```

Discussion
The value is `nil` if the value cannot be represented as a date.

Availability
Available in iOS 4.0 and later.

Declared In
`AVMetadataItem.h`

extraAttributes

Provides a dictionary of the additional attributes. (read-only)

```
@property(readonly, copy) NSDictionary *extraAttributes
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVMetadataItem.h`

key

Indicates the metadata item's key. (read-only)

```
@property(readonly, copy) id<NSObject, NSCopying> key
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVMetadataItem.h`

keySpace

Indicates the key space of metadata item's key. (read-only)

```
@property(readonly, copy) NSString *keySpace
```

Discussion
This is typically the default key space for the metadata container in which the metadata item is stored

Availability
Available in iOS 4.0 and later.

Declared In
`AVMetadataItem.h`

locale

Indicates the locale of the metadata item. (read-only)

```
@property(readonly, copy) NSLocale *locale
```

Discussion
The locale may be `nil` if no locale information is available for the metadata item.

Availability
Available in iOS 4.0 and later.

Declared In
`AVMetadataItem.h`

numberValue

Provides the value of the metadata item as a number. (read-only)

```
@property(readonly) NSNumber *numberValue
```

Discussion
The value is `nil` if the value cannot be represented as a number.

Availability
Available in iOS 4.0 and later.

Declared In
`AVMetadataItem.h`

stringValue

Provides the value of the metadata item as a string. (read-only)

```
@property(readonly) NSString *stringValue
```

Discussion
The value is `nil` if the value cannot be represented as a string.

Availability
Available in iOS 4.0 and later.

Declared In
`AVMetadataItem.h`

time

Indicates the timestamp of the metadata item. (read-only)

```
@property(readonly) CMTime time
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVMetadataItem.h`

value

Provides the value of the metadata item. (read-only)

```
@property(readonly, copy) id<NSObject, NSCopying> value
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVMetadataItem.h`

Class Methods

metadataItemsFromArray:withKey:keySpace:

Returns from a given array an array of metadata items that match a specified key or key space.

```
+ (NSArray *)metadataItemsFromArray:(NSArray *)array withKey:(id)key
    keySpace:(NSString *)keySpace
```

Parameters

array

An array of `AVMetadataItem` objects.

key

The key that must be matched for a metadata item to be included in the output array.

The key is compared to the keys in the metadata in the array using `isEqual:`.

If you don't want to filter by key, pass `nil`.

keySpace

The key space that must be matched for a metadata item to be included in the output array.

The key space is compared to the key spaces in the metadata in the array using `isEqualToString:`.

If you don't want to filter by key, pass `nil`.

Return Value
An array of the metadata items from *array* that match *key* or *keySpace*.

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVMetadataItem.h`

metadataItemsFromArray:withLocale:

Returns from a given array an array of metadata items that match a specified locale.

```
+ (NSArray *)metadataItemsFromArray:(NSArray *)array withLocale:(NSLocale *)locale
```

Parameters

array

An array of `AVMetadataItem` objects.

locale

The locale that must be matched for a metadata item to be included in the output array.

Return Value
An array of the metadata items from *array* that match *locale*.

Availability
Available in iOS 4.0 and later.

Declared In
`AVMetadataItem.h`

AVMutableAudioMix Class Reference

Inherits from	AVAudioMix : NSObject
Conforms to	NSCopying (AVAudioMix) NSMutableCopying (AVAudioMix) NSObject (NSObject)
Framework	/System/Library/Frameworks/AVFoundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	AVAudioMix.h

Overview

An `AVMutableAudioMix` object manages the input parameters for mixing audio tracks. It allows custom audio processing to be performed on audio tracks during playback or other operations.

Tasks

Creating a Mix

`+ audioMix` (page 186)
Returns a new mutable audio mix.

Input Parameters

`inputParameters` (page 186) *property*
The parameters for inputs to the mix

Properties

For more about Objective-C properties, see “Properties” in *The Objective-C Programming Language*.

inputParameters

The parameters for inputs to the mix

```
@property(nonatomic, copy) NSArray *inputParameters
```

Discussion
The array contains instances of `AVAudioMixInputParameters`. Note that an instance of `AVAudioMixInputParameters` is not required for each audio track that contributes to the mix; audio for those without associated `AVAudioMixInputParameters` will be included in the mix, processed according to default behavior.

Availability
Available in iOS 4.0 and later.

Declared In
`AVAudioMix.h`

Class Methods

audioMix

Returns a new mutable audio mix.

```
+ (AVMutableAudioMix *)audioMix
```

Return Value
A new mutable audio mix.

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVAudioMix.h`

AVMutableAudioMixInputParameters Class Reference

Inherits from	AVAudioMixInputParameters : NSObject
Conforms to	NSCopying (AVAudioMixInputParameters) NSMutableCopying (AVAudioMixInputParameters) NSObject (NSObject)
Framework	/System/Library/Frameworks/AVFoundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	AVAudioMix.h

Overview

An `AVMutableAudioMixInputParameters` object represents the parameters that should be applied to an audio track when it is added to a mix.

Tasks

Creating Input Parameters

`+ audioMixInputParameters` (page 188)
Returns a mutable input parameters object with no volume ramps and `trackID` (page 188) initialized to `kCMPersistentTrackID_Invalid`.

`+ audioMixInputParametersWithTrack:` (page 188)
Returns a mutable input parameters object for a given track.

Managing the Track ID

`trackID` (page 188) *property*
The trackID of the audio track to which the parameters should be applied.

Setting the Volume

`- setVolume:atTime:` (page 189)
Sets the value of the audio volume at a specific time.

`- setVolumeRampFromStartVolume:toEndVolume:timeRange:` (page 189)
Sets a volume ramp to apply during a specified time range.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C Programming Language*.

trackID

The trackID of the audio track to which the parameters should be applied.

```
@property(nonatomic) CMPersistentTrackID trackID
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVAudioMix.h`

Class Methods

audioMixInputParameters

Returns a mutable input parameters object with no volume ramps and `trackID` (page 188) initialized to `kCMPersistentTrackID_Invalid`.

```
+ (AVMutableAudioMixInputParameters *)audioMixInputParameters
```

Return Value
A mutable input parameters object with no volume ramps and `trackID` (page 188) initialized to `kCMPersistentTrackID_Invalid`.

Availability
Available in iOS 4.0 and later.

Declared In
`AVAudioMix.h`

audioMixInputParametersWithTrack:

Returns a mutable input parameters object for a given track.

```
+ (AVMutableAudioMixInputParameters *)audioMixInputParametersWithTrack:(AVAssetTrack
    *)track
```

Parameters

track

The track for which to create input parameters.

Return Value

A mutable input parameters object with no volume ramps and `trackID` (page 188) set to *track*'s trackID.

Discussion

Availability

Available in iOS 4.0 and later.

Declared In

`AVAudioMix.h`

Instance Methods

setVolume:atTime:

Sets the value of the audio volume at a specific time.

```
- (void)setVolume:(float)volume atTime:(CMTime)time
```

Parameters

volume

The volume.

time

The time at which to set the volume to *volume*.

Discussion

Availability

Available in iOS 4.0 and later.

Declared In

`AVAudioMix.h`

setVolumeRampFromStartVolume:toEndVolume:timeRange:

Sets a volume ramp to apply during a specified time range.

```
- (void)setVolumeRampFromStartVolume:(float)startVolume toEndVolume:(float)endVolume
    timeRange:(CMTimeRange)timeRange
```

Parameters

startVolume

The starting volume.

endVolume

The end volume.

timeRange
The time range over which to apply the ramp.

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
AVAudioMix.h

AVMutableComposition Class Reference

Inherits from	AVComposition : AVAsset : NSObject
Conforms to	NSMutableCopying (AVComposition) NSCopying (AVAsset) AVAsynchronousKeyValueLoading (AVAsset) NSObject (NSObject)
Framework	/System/Library/Frameworks/AVFoundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	AVComposition.h

Overview

`AVMutableComposition` is a mutable subclass of `AVComposition` you use when you want to create a new composition from existing assets. You can add and remove tracks, and you can add, remove, and scale time ranges.

You can make an immutable snapshot of a mutable composition for playback or inspection as follows:

```
AVMutableComposition *myMutableComposition =
    <#a mutable composition you want to inspect or play in its current state#>;

AVComposition *immutableSnapshotOfMyComposition = [myMutableComposition copy];

// Create a player to inspect and play the composition.
AVPlayerItem *playerItemForSnapshottedComposition =
    [[AVPlayerItem alloc] initWithAsset:immutableSnapshotOfMyComposition];
```

Tasks

Managing Time Ranges

- `insertEmptyTimeRange:` (page 194)
  Adds or extends an empty timeRange within all tracks of the composition.
- `insertTimeRange:ofAsset:atTime:error:` (page 194)
  Inserts all the tracks within a given time range of a specified asset into the receiver.

- `removeTimeRange:` (page 196)
  Removes a specified timeRange from all tracks of the composition.
- `scaleTimeRange:toDuration:` (page 197)
  Changes the duration of all tracks in a given time range.

Creating a Mutable Composition

+ `composition` (page 193)
  Returns a new, empty, mutable composition.

Managing Tracks

`tracks` (page 193) *property*
  An array of `AVMutableCompositionTrack` objects contained by the composition. (read-only)

- `addMutableTrackWithMediaType:preferredTrackID:` (page 193)
  Adds an empty track to the receiver.
- `removeTrack:` (page 196)
  Removes a specified track from the receiver.
- `mutableTrackCompatibleWithTrack:` (page 195)
  Returns a track in the receiver into which any time range of a given asset track can be inserted.

Video Size

`naturalSize` (page 192) *property*
  The encoded or authored size of the visual portion of the asset.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C Programming Language*.

naturalSize

The encoded or authored size of the visual portion of the asset.

```
@property(nonatomic) CGSize naturalSize
```

Discussion
If this value is not set, the default behavior is as defined by `AVAsset`; set the value to `CGSizeZero` to revert to the default behavior.

Availability
Available in iOS 4.0 and later.

Declared In
`AVComposition.h`

tracks

An array of `AVMutableCompositionTrack` objects contained by the composition. (read-only)

```
@property(nonatomic, readonly) NSArray *tracks
```

Discussion
In a mutable composition, the tracks are instances of `AVMutableCompositionTrack`, whereas in `AVComposition` the tracks are instances of `AVCompositionTrack`.

Availability
Available in iOS 4.0 and later.

Declared In
`AVComposition.h`

Class Methods

composition

Returns a new, empty, mutable composition.

```
+ (AVMutableComposition *)composition
```

Return Value
A new, empty, mutable composition.

Availability
Available in iOS 4.0 and later.

Declared In
`AVComposition.h`

Instance Methods

addMutableTrackWithMediaType:preferredTrackID:

Adds an empty track to the receiver.

```
- (AVMutableCompositionTrack *)addMutableTrackWithMediaType:(NSString *)mediaType
    preferredTrackID:(CMPersistentTrackID)preferredTrackID
```

Parameters
mediaType
The media type of the new track.

preferredTrackID

The preferred track ID for the new track. If you do not need to specify a preferred track ID, pass `kCMPersistentTrackID_Invalid`.

The preferred track ID will be used for the new track provided that it is not currently in use and has not previously been used. If the preferred track ID you specify is not available, or if you pass in `kCMPersistentTrackID_Invalid`, a unique track ID is generated.

Return Value
An instance of `AVMutableCompositionTrack` representing the new track.

Discussion
You can get the actual trackID of the new track through its `@"trackID"` key.

Availability
Available in iOS 4.0 and later.

See Also
`- mutableTrackCompatibleWithTrack:` (page 195)

Declared In
`AVComposition.h`

insertEmptyTimeRange:

Adds or extends an empty timeRange within all tracks of the composition.

```
- (void)insertEmptyTimeRange:(CMTimeRange)timeRange
```

Parameters
timeRange

The empty time range to insert.

Discussion
If you insert an empty time range into the composition, any media that was presented during that interval prior to the insertion will be presented instead immediately afterward. You can use this method to reserve an interval in which you want a subsequently created track to present its media.

Availability
Available in iOS 4.0 and later.

See Also
`- insertTimeRange:ofAsset:atTime:error:` (page 194)

Declared In
`AVComposition.h`

insertTimeRange:ofAsset:atTime:error:

Inserts all the tracks within a given time range of a specified asset into the receiver.

```
- (BOOL)insertTimeRange:(CMTimeRange)timeRange ofAsset:(AVAsset *)asset
    atTime:(CMTime)startTime error:(NSError **)outError
```

Parameters

timeRange

The time range of the asset to be inserted.

asset

An asset that contains the tracks to be inserted.

startTime

The time at which the inserted tracks should be presented by the receiver.

outError

If the insertion was not successful, on return contains an `NSError` object that describes the problem.

Return Value

`YES` if the insertion was successful, otherwise `NO`.

Discussion

This method may add new tracks to ensure that all tracks of the asset are represented in the inserted time range.

Existing content at the specified start time is pushed out by the duration of the time range.

Media data for the inserted time range is presented at its natural duration; you can scale it to a different duration using `scaleTimeRange:toDuration:` (page 197).

Availability

Available in iOS 4.0 and later.

See Also

`- insertEmptyTimeRange:` (page 194)

Declared In

`AVComposition.h`

mutableTrackCompatibleWithTrack:

Returns a track in the receiver into which any time range of a given asset track can be inserted.

```
- (AVMutableCompositionTrack *)mutableTrackCompatibleWithTrack:(AVAssetTrack *)track
```

Parameters

track

An AVAssetTrack from which a time range may be inserted.

Return Value

A mutable track in the receiver into which any time range of *track* can be inserted. If no such track is available, the returns `nil`.

Discussion

For best performance, you should keep the number of tracks of a composition should be kept to a minimum, corresponding to the number for which media data must be presented in parallel. If you want to present media data of the same type serially, even from multiple assets, you should use a single track of that media type. You use this method to identify a suitable existing target track for an insertion.

If there is no compatible track available, you can create a new track of the same media type as *track* using `addMutableTrackWithMediaType:preferredTrackID:` (page 193).

This method is similar to `compatibleTrackForCompositionTrack:` (page 263) (`AVAsset`).

Availability
Available in iOS 4.0 and later.

See Also
`- addMutableTrackWithMediaType:preferredTrackID:` (page 193)

Declared In
`AVComposition.h`

removeTimeRange:

Removes a specified timeRange from all tracks of the composition.

```
- (void)removeTimeRange:(CMTimeRange)timeRange
```

Parameters

timeRange
The time range to be removed.

Discussion
After removing, existing content after the time range will be pulled in.

Removal of a time range does not cause any existing tracks to be removed from the composition, even if removing *timeRange* results in an empty track. Instead, it removes or truncates track segments that intersect with the time range.

Availability
Available in iOS 4.0 and later.

See Also
`- removeTrack:` (page 196)

Declared In
`AVComposition.h`

removeTrack:

Removes a specified track from the receiver.

```
- (void)removeTrack:(AVCompositionTrack *)track
```

Parameters

track
The track to remove.

Discussion
When it is removed *track*'s `@"composition"` key is set to `nil`. The values of its other keys remain intact, for arbitrary use.

Availability
Available in iOS 4.0 and later.

See Also
- `removeTimeRange:` (page 196)

Declared In
`AVComposition.h`

scaleTimeRange:toDuration:

Changes the duration of all tracks in a given time range.

```
- (void)scaleTimeRange:(CMTimeRange)timeRange toDuration:(CMTime)duration
```

Parameters

timeRange
The time range of the composition to be scaled.

duration
The new duration of *timeRange*.

Discussion
Each track segment affected by the scaling operation will be presented at a rate equal to `source.duration / target.duration` of its resulting time mapping.

Availability
Available in iOS 4.0 and later.

Declared In
`AVComposition.h`

AVMutableCompositionTrack Class Reference

Inherits from	AVCompositionTrack : AVAssetTrack : NSObject
Conforms to	NSCopying (AVAssetTrack) AVAsynchronousKeyValueLoading (AVAssetTrack) NSObject (NSObject)
Framework	/System/Library/Frameworks/AVFoundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	AVCompositionTrack.h

Overview

`AVMutableCompositionTrack` is a mutable subclass of `AVCompositionTrack` that lets you for insert, remove, and scale track segments without affecting their low-level representation (that is, the operations you perform are non-destructive on the original).

`AVCompositionTrack` defines constraints for the temporal alignment of the track segments. If you set the array of track segments in a mutable composition (see `trackSegments` (page 202)), you can test whether the segments meet the constraints using `validateTrackSegments:error:` (page 204).

Tasks

Managing Time Ranges

`- insertEmptyTimeRange:` (page 202)
Adds or extends an empty time range within the receiver.

`- insertTimeRange:ofTrack:atTime:error:` (page 203)
Inserts a time range of a source track.

`- removeTimeRange:` (page 203)
Removes a specified time range from the receiver.

`- scaleTimeRange:toDuration:` (page 204)
Changes the duration of a time range in the receiver.

`segments` (page 202) *property*
The composition track's array of track segments.

Validating Segments

`- validateTrackSegments:error:` (page 204)
Returns a Boolean value that indicates whether a given array of track segments conform to the timing rules for a composition track.

Track Properties

`languageCode` (page 200) *property*
The language associated with the track, as an ISO 639-2/T language code.

`extendedLanguageTag` (page 200) *property*
The language tag associated with the track, as an RFC 4646 language tag.

`naturalTimeScale` (page 201) *property*
The timescale in which time values for the track can be operated upon without extraneous numerical conversion.

`preferredTransform` (page 201) *property*
The preferred transformation of the visual media data for display purposes.

`preferredVolume` (page 201) *property*
The preferred volume of the audible media data.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C Programming Language*.

extendedLanguageTag

The language tag associated with the track, as an RFC 4646 language tag.

```
@property(nonatomic, copy) NSString *extendedLanguageTag
```

Discussion
If not set, the value is `nil`.

Availability
Available in iOS 4.0 and later.

See Also
`@property languageCode` (page 200)

Declared In
`AVCompositionTrack.h`

languageCode

The language associated with the track, as an ISO 639-2/T language code.

```
@property(nonatomic, copy) NSString *languageCode
```

Discussion
If not set, the value is `nil`.

Availability
Available in iOS 4.0 and later.

See Also
`@property extendedLanguageTag` (page 200)

Declared In
`AVCompositionTrack.h`

naturalTimeScale

The timescale in which time values for the track can be operated upon without extraneous numerical conversion.

```
@property(nonatomic) CMTimeScale naturalTimeScale
```

Discussion
If not set, the value is the natural time scale of the first non-empty edit, or 600 if there are no non-empty edits.

Set the value to `0` to revert to the default behavior.

Availability
Available in iOS 4.0 and later.

Declared In
`AVCompositionTrack.h`

preferredTransform

The preferred transformation of the visual media data for display purposes.

```
@property(nonatomic) CGAffineTransform preferredTransform
```

Discussion
If not set, the value is `CGAffineTransformIdentity`.

Availability
Available in iOS 4.0 and later.

Declared In
`AVCompositionTrack.h`

preferredVolume

The preferred volume of the audible media data.

```
@property(nonatomic) float preferredVolume
```

Discussion
If not set, the value is `1.0`.

Availability
Available in iOS 4.0 and later.

Declared In
`AVCompositionTrack.h`

segments

The composition track's array of track segments.

```
@property(nonatomic, copy) NSArray *segments
```

Special Considerations

The `timeMapping.target.start` of the first track segment must be `kCMTimeZero`, and the `timeMapping.target.start` of each subsequent track segment must equal `CMTimeRangeGetEnd(<#previousTrackSegment#>.timeMapping.target)`. You can use `validateTrackSegments:error:` (page 204) to ensure that an array of track segments conforms to this rule.

Availability
Available in iOS 4.0 and later.

Declared In
`AVCompositionTrack.h`

Instance Methods

insertEmptyTimeRange:

Adds or extends an empty time range within the receiver.

```
- (void)insertEmptyTimeRange:(CMTimeRange)timeRange
```

Parameters

timeRange

The empty time range to be inserted.

Discussion
If you insert an empty time range into the track, any media that was presented during that interval prior to the insertion will be presented instead immediately afterward.

The nature of the data inserted depends upon the media type of the track. For example, an empty time range in a sound track presents silence.

Availability
Available in iOS 4.0 and later.

Declared In
AVCompositionTrack.h

insertTimeRange:ofTrack:atTime:error:

Inserts a time range of a source track.

```
- (BOOL)insertTimeRange:(CMTimeRange)timeRange ofTrack:(AVAssetTrack *)track
    atTime:(CMTime)startTime error:(NSError **)error
```

Parameters

timeRange
The time range of the track to be inserted.

track
The source track to be inserted.

startTime
The time at which *track* is to be presented by the composition track.

error
If *track* is not inserted successfully, contains an NSError object that describes the problem.

Return Value
YES if *track* was inserted successfully, otherwise NO.

Discussion
By default, the inserted track's time range is presented at its natural duration and rate. You can scale it to a different duration (so that it is presented at a different rate) using scaleTimeRange:toDuration: (page 204).

Insertion might fail if, for example, the asset that you try to insert is restricted by copy-protection.

Availability
Available in iOS 4.0 and later.

Declared In
AVCompositionTrack.h

removeTimeRange:

Removes a specified time range from the receiver.

```
- (void)removeTimeRange:(CMTimeRange)timeRange
```

Parameters

timeRange
The time range to be removed.

Discussion
Removing a time range does not cause the track to be removed from the composition. Instead it removes or truncates track segments that intersect with the time range.

Availability
Available in iOS 4.0 and later.

Declared In
`AVCompositionTrack.h`

scaleTimeRange:toDuration:

Changes the duration of a time range in the receiver.

```
- (void)scaleTimeRange:(CMTimeRange)timeRange toDuration:(CMTime)duration
```

Parameters

timeRange
The time range of the track to be scaled.

duration
The new duration of *timeRange*.

Discussion
Each track segment affected by the scaling operation will be presented at a rate equal to `source.duration / target.duration` of its resulting `timeMapping`.

Availability
Available in iOS 4.0 and later.

Declared In
`AVCompositionTrack.h`

validateTrackSegments:error:

Returns a Boolean value that indicates whether a given array of track segments conform to the timing rules for a composition track.

```
- (BOOL)validateTrackSegments:(NSArray *)trackSegments error:(NSError **)error
```

Parameters

trackSegments
An array of `AVCompositionTrackSegment` objects.

error
If validation fails, on return contains an `NSError` object that describes the problem.

Return Value
`YES` if the track segments in *trackSegments* conform to the timing rules for a composition track, otherwise `NO`.

Discussion
You can use this method to ensure that an array of track segments is suitable for setting as the value of the `trackSegments` (page 202) property. The `timeMapping.target.start` of the first track segment must be `kCMTimeZero`, and the `timeMapping.target.start` of each subsequent track segment must equal `CMTimeRangeGetEnd(<#previousTrackSegment#>.timeMapping.target)`.

If you want to modify the existing `trackSegments` (page 202) array, you can create a mutable copy of it, modify the mutable array, and then validate the mutable array using this method.

Availability
Available in iOS 4.0 and later.

Declared In
`AVCompositionTrack.h`

AVMutableMetadataItem Class Reference

Inherits from	AVMetadataItem : NSObject
Conforms to	NSCopying (AVMetadataItem) NSMutableCopying (AVMetadataItem) NSObject (NSObject)
Framework	/System/Library/Frameworks/AVFoundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	AVMetadataItem.h

Overview

`AVMutableMetadataItem` is a mutable subclass of `AVMetadataItem` that lets you build collections of metadata to be written to asset files using `AVAssetExportSession`.

You can initialize a mutable metadata item from an existing `AVMetadataItem` object or with a one or more of the basic properties of a metadata item: a key, a key space, a locale, and a value.

Tasks

Creating a Mutable Metadata Item

`+ metadataItem` (page 210)
Returns a new mutable metadata item.

Key and Key Space

`key` (page 208) *property*
Indicates the metadata item's key.

`keySpace` (page 208) *property*
Indicates the key space of the metadata item's key.

Values

`value` (page 209) *property*
Indicates the metadata item's value.

`locale` (page 209) *property*
Indicates the metadata item's locale.

`time` (page 209) *property*
Indicates the metadata item's timestamp.

`extraAttributes` (page 208) *property*
Provides a dictionary of the metadata item's additional attributes.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C Programming Language*.

extraAttributes

Provides a dictionary of the metadata item's additional attributes.

```
@property(readwrite, copy) NSDictionary *extraAttributes
```

Availability
Available in iOS 4.0 and later.

Declared In
`AVMetadataItem.h`

key

Indicates the metadata item's key.

```
@property(readwrite, copy) id<NSObject, NSCopying> key
```

Availability
Available in iOS 4.0 and later.

Declared In
`AVMetadataItem.h`

keySpace

Indicates the key space of the metadata item's key.

```
@property(readwrite, copy) NSString *keySpace
```

Discussion
This is typically the default key space for the metadata container in which the metadata item is stored.

Availability
Available in iOS 4.0 and later.

Declared In
`AVMetadataItem.h`

locale

Indicates the metadata item's locale.

```
@property(readwrite, copy) NSLocale *locale
```

Discussion
The locale may be `nil` if no locale information is available for the item.

Availability
Available in iOS 4.0 and later.

Declared In
`AVMetadataItem.h`

time

Indicates the metadata item's timestamp.

```
@property(readwrite) CMTime time
```

Availability
Available in iOS 4.0 and later.

Declared In
`AVMetadataItem.h`

value

Indicates the metadata item's value.

```
@property(readwrite, copy) id<NSObject, NSCopying> value
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVMetadataItem.h`

Class Methods

metadataItem

Returns a new mutable metadata item.

```
+ (AVMutableMetadataItem *)metadataItem
```

Return Value
A new mutable metadata item.

Availability
Available in iOS 4.0 and later.

Declared In
`AVMetadataItem.h`

AVMutableVideoComposition Class Reference

Inherits from	AVVideoComposition : NSObject
Conforms to	NSCopying (AVVideoComposition) NSMutableCopying (AVVideoComposition) NSObject (NSObject)
Framework	/System/Library/Frameworks/AVFoundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	AVVideoComposition.h

Overview

An `AVMutableVideoComposition` object represents a mutable video composition.

Tasks

Creating a Video Composition

`+ videoComposition` (page 213)
Returns a new mutable video composition.

Properties

`frameDuration` (page 212) *property*
The interval for which the video composition should render composed video frames.

`renderSize` (page 213) *property*
The size at which the video composition should render.

`renderScale` (page 212) *property*
The scale at which the video composition should render.

`instructions` (page 212) *property*
The video composition instructions.

`animationTool` (page 212) *property*
A special video composition tool for use with Core Animation.

Properties

For more about Objective-C properties, see “Properties” in *The Objective-C Programming Language*.

animationTool

A special video composition tool for use with Core Animation.

```
@property(nonatomic, retain) AVVideoCompositionCoreAnimationTool *animationTool
```

Discussion
This attribute may be `nil`.

Availability
Available in iOS 4.0 and later.

Declared In
`AVVideoComposition.h`

frameDuration

The interval for which the video composition should render composed video frames.

```
@property(nonatomic) CMTime frameDuration
```

Availability
Available in iOS 4.0 and later.

Declared In
`AVVideoComposition.h`

instructions

The video composition instructions.

```
@property(nonatomic, copy) NSArray *instructions
```

Discussion
The array contains of instances of `AVVideoCompositionInstruction`.

Availability
Available in iOS 4.0 and later.

Declared In
`AVVideoComposition.h`

renderScale

The scale at which the video composition should render.

```
@property(nonatomic) float renderScale
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVVideoComposition.h`

renderSize

The size at which the video composition should render.

```
@property(nonatomic) CGSize renderSize
```

Availability
Available in iOS 4.0 and later.

Declared In
`AVVideoComposition.h`

Class Methods

videoComposition

Returns a new mutable video composition.

```
+ (AVMutableVideoComposition *)videoComposition
```

Return Value
A new mutable video composition.

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVVideoComposition.h`

AVMutableVideoCompositionInstruction Class Reference

Inherits from	AVVideoCompositionInstruction : NSObject
Conforms to	NSCoding (AVVideoCompositionInstruction) NSCopying (AVVideoCompositionInstruction) NSMutableCopying (AVVideoCompositionInstruction) NSObject (NSObject)
Framework	/System/Library/Frameworks/AVFoundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	AVVideoComposition.h

Overview

An `AVMutableVideoCompositionInstruction` object represents an operation to be performed by a compositor.

An `AVVideoComposition` object maintains an array of `instructions` to perform its composition.

Tasks

Creating an Instruction

`+ videoCompositionInstruction` (page 217)
Returns a new mutable video composition instruction.

Properties

`backgroundColor` (page 216) *property*
The background color of the composition.

`layerInstructions` (page 216) *property*
An array of instances of `AVVideoCompositionLayerInstruction` that specify how video frames from source tracks should be layered and composed.

`timeRange` (page 217) *property*
The time range during which the instruction is effective.

`enablePostProcessing` (page 216) *property*
Indicates whether post-processing should be allowed for the duration of the instruction.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C Programming Language*.

backgroundColor

The background color of the composition.

```
@property(nonatomic, retain) CGColorRef backgroundColor
```

Discussion
Only solid BGRA colors are supported; patterns and other color refs that are not supported are ignored. If the rendered pixel buffer does not have alpha, the alpha value of the background color is ignored.

If the background color is not specified, the video compositor will use a default background color of opaque black.

Availability
Available in iOS 4.0 and later.

Declared In
`AVVideoComposition.h`

enablePostProcessing

Indicates whether post-processing should be allowed for the duration of the instruction.

```
@property(nonatomic, assign) BOOL enablePostProcessing
```

Discussion
`NO` indicates that post-processing should be skipped for the duration of this instruction.

The value is `YES` by default.

Availability
Available in iOS 4.0 and later.

Declared In
`AVVideoComposition.h`

layerInstructions

An array of instances of `AVVideoCompositionLayerInstruction` that specify how video frames from source tracks should be layered and composed.

```
@property(nonatomic, copy) NSArray *layerInstructions
```

Discussion
Tracks are layered in the composition according to the top-to-bottom order of the `layerInstructions` array; the track with trackID of the first instruction in the array will be layered on top, with the track with the trackID of the second instruction immediately underneath, and so on.

If this key is `nil`, the output will be a fill of the background color.

Availability
Available in iOS 4.0 and later.

See Also
`@property backgroundColor` (page 216)

Declared In
`AVVideoComposition.h`

timeRange

The time range during which the instruction is effective.

```
@property(nonatomic, assign) CMTimeRange timeRange
```

Discussion
If the time range is invalid, the video compositor will ignore it.

Availability
Available in iOS 4.0 and later.

Declared In
`AVVideoComposition.h`

Class Methods

videoCompositionInstruction

Returns a new mutable video composition instruction.

```
+ (AVMutableVideoCompositionInstruction *)videoCompositionInstruction
```

Return Value
A new mutable video composition instruction.

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVVideoComposition.h`

AVMutableVideoCompositionLayerInstruction Class Reference

Inherits from	AVVideoCompositionLayerInstruction : NSObject
Conforms to	NSCoding (AVVideoCompositionLayerInstruction) NSCopying (AVVideoCompositionLayerInstruction) NSMutableCopying (AVVideoCompositionLayerInstruction) NSObject (NSObject)
Framework	/System/Library/Frameworks/AVFoundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	AVVideoComposition.h

Overview

`AVMutableVideoCompositionLayerInstruction` is a mutable subclass of `AVVideoCompositionLayerInstruction` that you use to modify the transform and opacity ramps to apply to a given track in an AV composition.

Tasks

Creating an Instruction

`+ videoCompositionLayerInstruction` (page 220)
Returns a new mutable video composition layer instruction.

`+ videoCompositionLayerInstructionWithAssetTrack:` (page 221)
Returns a new mutable video composition layer instruction for the given track.

Track ID

`trackID` (page 220) *property*
The trackID of the source track to which the compositor will apply the instruction.

Managing Properties

- `setOpacity:atTime:` (page 221)
  Sets a value of the opacity at a time within the time range of the instruction.
- `setOpacityRampFromStartOpacity:toEndOpacity:timeRange:` (page 221)
  Sets an opacity ramp to apply during a specified time range.
- `setTransform:atTime:` (page 222)
  Sets a value of the transform at a time within the time range of the instruction.
- `setTransformRampFromStartTransform:toEndTransform:timeRange:` (page 222)
  Sets a transform ramp to apply during a given time range.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C Programming Language*.

trackID

The trackID of the source track to which the compositor will apply the instruction.

```
@property(nonatomic, assign) CMPersistentTrackID trackID
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVVideoComposition.h`

Class Methods

videoCompositionLayerInstruction

Returns a new mutable video composition layer instruction.

```
+ (AVMutableVideoCompositionLayerInstruction *)videoCompositionLayerInstruction
```

Return Value
A new mutable video composition layer instruction with no transform or opacity ramps and `trackID` (page 220) initialized to `kCMPersistentTrackID_Invalid`.

Availability
Available in iOS 4.0 and later.

Declared In
`AVVideoComposition.h`

videoCompositionLayerInstructionWithAssetTrack:

Returns a new mutable video composition layer instruction for the given track.

```
+ (AVMutableVideoCompositionLayerInstruction
    *)videoCompositionLayerInstructionWithAssetTrack:(AVAssetTrack *)track
```

Parameters

track

The asset track to which to apply the instruction.

Return Value

A new mutable video composition layer instruction with no transform or opacity ramps and `trackID` (page 220) initialized to the track ID of *track*.

Discussion

Availability

Available in iOS 4.0 and later.

Declared In

AVVideoComposition.h

Instance Methods

setOpacity:atTime:

Sets a value of the opacity at a time within the time range of the instruction.

```
- (void)setOpacity:(float)opacity atTime:(CMTime)time
```

Parameters

opacity

The opacity to be applied at *time*. The value must be between 0.0 and 1.0.

time

A time value within the time range of the composition instruction.

Discussion

Sets a fixed opacity to apply from the specified time until the next time at which an opacity is set; this is the same as setting a flat ramp for that time range. Before the first time for which an opacity is set, the opacity is held constant at 1.0; after the last specified time, the opacity is held constant at the last value.

Availability

Available in iOS 4.0 and later.

Declared In

AVVideoComposition.h

setOpacityRampFromStartOpacity:toEndOpacity:timeRange:

Sets an opacity ramp to apply during a specified time range.

```
- (void)setOpacityRampFromStartOpacity:(float)startOpacity
    toEndOpacity:(float)endOpacity timeRange:(CMTimeRange)timeRange
```

Parameters

startOpacity

The opacity to be applied at the start time of *timeRange*. The value must be between 0.0 and 1.0.

endOpacity

The opacity to be applied at the end time of *timeRange*. The value must be between 0.0 and 1.0.

timeRange

The time range over which the value of the opacity will be interpolated between *startOpacity* and *endOpacity*.

Discussion

During an opacity ramp, opacity is computed using a linear interpolation. Before the first time for which an opacity is set, the opacity is held constant at 1.0; after the last specified time, the opacity is held constant at the last value.

Availability

Available in iOS 4.0 and later.

Declared In

`AVVideoComposition.h`

setTransform:atTime:

Sets a value of the transform at a time within the time range of the instruction.

```
- (void)setTransform:(CGAffineTransform)transform atTime:(CMTime)time
```

Parameters

transform

The transform to be applied at *time*.

time

A time value within the time range of the composition instruction.

Discussion

Sets a fixed transform to apply from the specified time until the next time at which a transform is set. This is the same as setting a flat ramp for that time range. Before the first specified time for which a transform is set, the affine transform is held constant at the value of `CGAffineTransformIdentity`; after the last time for which a transform is set, the affine transform is held constant at that last value.

Availability

Available in iOS 4.0 and later.

Declared In

`AVVideoComposition.h`

setTransformRampFromStartTransform:toEndTransform:timeRange:

Sets a transform ramp to apply during a given time range.

```
- (void)setTransformRampFromStartTransform:(CGAffineTransform)startTransform
    toEndTransform:(CGAffineTransform)endTransform timeRange:(CMTimeRange)timeRange
```

Parameters

startTransform

The transform to be applied at the starting time of *timeRange*.

endTransform

The transform to be applied at the end time of *timeRange*.

timeRange

The time range over which the value of the transform will be interpolated between *startTransform* and *endTransform*.

Discussion

During a transform ramp, the affine transform is interpolated between the values set at the ramp's start time and end time. Before the first specified time for which a transform is set, the affine transform is held constant at the value of `CGAffineTransformIdentity`; after the last time for which a transform is set, the affine transform is held constant at that last value.

Availability

Available in iOS 4.0 and later.

Declared In

`AVVideoComposition.h`

AVPlayer Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/AVFoundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	AVPlayer.h

Overview

An `AVPlayer` object offers a playback interface for single- or multiple-item playback that you use to implement playback controllers and playback user interfaces. The multiple-item case supports advanced behaviors.

A player works equally well with local and remote media files, providing you with appropriate information about readiness to play, or about the need to wait for additional data before continuing.

You can configure a player to display visual media to CoreAnimation layers, or to vend images for processing, or both simultaneously. The player also supports selection of item tracks according to language preference.

Tasks

Creating a Player

`- initWithURL:` (page 232)
Initializes a new player to play a single audiovisual resource referenced by a given URL.

`+ playerWithURL:` (page 229)
Returns a new player to play a single audiovisual resource referenced by a given URL.

`- initWithPlayerItem:` (page 232)
Initializes a new player to play a given single audiovisual item.

`+ playerWithPlayerItem:` (page 229)
Returns a new player initialized to play a given single audiovisual item

Managing Playback

- `play` (page 233)
  Begins playback of the current item.
- `pause` (page 233)
  Pauses playback.

`rate` (page 228) *property*
The current rate of playback.

`actionAtItemEnd` (page 227) *property*
The action to perform when an item has finished playing.

- `replaceCurrentItemWithPlayerItem:` (page 234)
  Replaces the player item with a new player item.

Managing Time

- `currentTime` (page 232)
  Returns the current time of the current item.
- `seekToTime:` (page 235)
  Moves the playback cursor to a given time.
- `seekToTime:toleranceBefore:toleranceAfter:` (page 235)
  Moves the playback cursor within a specified time bound.

Timed Observations

- `addPeriodicTimeObserverForInterval:queue:usingBlock:` (page 231)
  Requests invocation of a given block during playback to report changing time.
- `addBoundaryTimeObserverForTimes:queue:usingBlock:` (page 230)
  Requests invocation of a block when specified times are traversed during normal playback.
- `removeTimeObserver:` (page 234)
  Cancels a previously registered time observer.

Configuring a Player

`closedCaptionDisplayEnabled` (page 227) *property*
Indicates whether the player uses closed captioning.

Player Properties

`status` (page 228) *property*
Indicates whether the player can be used for playback. (read-only)

`error` (page 228) *property*
If the receiver's status is `AVPlayerStatusFailed` (page 236), this describes the error that caused the failure. (read-only)

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C Programming Language*.

actionAtItemEnd

The action to perform when an item has finished playing.

```
@property(nonatomic) AVPlayerActionAtItemEnd actionAtItemEnd
```

Discussion
For possible values, see "AVPlayerActionAtItemEnd" (page 236).

Availability
Available in iOS 4.0 and later.

Declared In
`AVPlayer.h`

closedCaptionDisplayEnabled

Indicates whether the player uses closed captioning.

```
@property(nonatomic, getter=isClosedCaptionDisplayEnabled) BOOL
    closedCaptionDisplayEnabled
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVPlayer.h`

currentItem

The player's current item. (read-only)

```
@property(nonatomic, readonly) AVPlayerItem *currentItem
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVPlayer.h`

error

If the receiver's status is `AVPlayerStatusFailed` (page 236), this describes the error that caused the failure. (read-only)

```
@property(nonatomic, readonly) NSError *error
```

Discussion
The value of this property is an error object that describes what caused the receiver to no longer be able to play items. If the receiver's status is not `AVPlayerStatusFailed` (page 236), the value of this property is `nil`.

Availability
Available in iOS 4.0 and later.

Declared In
`AVPlayer.h`

rate

The current rate of playback.

```
@property(nonatomic) float rate
```

Discussion
`0.0` means “stopped”, `1.0` means “play at the natural rate of the current item”.

Availability
Available in iOS 4.0 and later.

See Also
- `play` (page 233)
- `pause` (page 233)

Declared In
`AVPlayer.h`

status

Indicates whether the player can be used for playback. (read-only)

```
@property(nonatomic, readonly) AVPlayerStatus status
```

Discussion
When the value of this property is `AVPlayerStatusFailed` (page 236), you can no longer use the player for playback and you need to create a new instance to replace it. If this happens, you can check the value of the error property to determine the nature of the failure.

This property is key value observable.

Availability
Available in iOS 4.0 and later.

Declared In
`AVPlayer.h`

Class Methods

playerWithPlayerItem:

Returns a new player initialized to play a given single audiovisual item

```
+ (AVPlayer *)playerWithPlayerItem:(AVPlayerItem *)item
```

Parameters

item

A player item.

Return Value
A new player, initialized to play *item*.

Discussion
You can use this method to play items for which an `AVAsset` object has previously been created (see `initWithAsset:` (page 247) in `AVPlayerItem`).

Availability
Available in iOS 4.0 and later.

See Also
`- initWithPlayerItem:` (page 232)

Declared In
`AVPlayer.h`

playerWithURL:

Returns a new player to play a single audiovisual resource referenced by a given URL.

```
+ (AVPlayer *)playerWithURL:(NSURL *)URL
```

Parameters

URL

An URL that identifies an audiovisual resource.

Return Value
A new player initialized to play the audiovisual resource specified by *URL*.

Discussion
This method implicitly creates an `AVPlayerItem` object. You can get the player item using `currentItem` (page 227).

Availability
Available in iOS 4.0 and later.

See Also
- `initWithURL:` (page 232)
- `@property currentItem` (page 227)

Declared In
`AVPlayer.h`

Instance Methods

addBoundaryTimeObserverForTimes:queue:usingBlock:

Requests invocation of a block when specified times are traversed during normal playback.

```
- (id)addBoundaryTimeObserverForTimes:(NSArray *)times queue:(dispatch_queue_t)queue
    usingBlock:(void (^)(void))block
```

Parameters

times

An array of `NSValue` objects containing `CMTime` values representing the times at which to invoke *block*.

queue

A serial queue onto which *block* should be enqueued.

If you pass `NULL`, the main queue (obtained using `dispatch_get_main_queue`) is used. Passing a concurrent queue will result in undefined behavior.

block

The block to be invoked when any of the times in *times* is crossed during normal playback.

Return Value
An opaque object.

Discussion
You must retain the returned value as long as you want the time observer to be invoked by the player. Each invocation of this method should be paired with a corresponding call to `removeTimeObserver:` (page 234).

Special Considerations

The thread *block* is invoked on may not be serviced by an application run loop. If you need to perform an operation in the user interface, you must ensure that the work is bounced to the main thread.

Availability
Available in iOS 4.0 and later.

See Also
- `addPeriodicTimeObserverForInterval:queue:usingBlock:` (page 231)
- `removeTimeObserver:` (page 234)
- `@property currentTime` (page 232)

Declared In
`AVPlayer.h`

addPeriodicTimeObserverForInterval:queue:usingBlock:

Requests invocation of a given block during playback to report changing time.

```
- (id)addPeriodicTimeObserverForInterval:(CMTime)interval
    queue:(dispatch_queue_t)queue usingBlock:(void (^)(CMTime time))block
```

Parameters

interval

The interval of invocation of the block during normal playback, according to progress of the current time of the player.

queue

A serial queue onto which *block* should be enqueued.

If you pass `NULL`, the main queue (obtained using `dispatch_get_main_queue`) is used. Passing a concurrent queue will result in undefined behavior.

block

The block to be invoked periodically.

The block takes a single parameter:

time

The time at which the block is invoked.

Return Value

An opaque object.

Discussion

You must retain the returned value as long as you want the time observer to be invoked by the player. Each invocation of this method should be paired with a corresponding call to `removeTimeObserver:` (page 234).

The block is invoked periodically at the interval specified, interpreted according to the timeline of the current item. The block is also invoked whenever time jumps and whenever playback starts or stops. If the interval corresponds to a very short interval in real time, the player may invoke the block less frequently than requested. Even so, the player will invoke the block sufficiently often for the client to update indications of the current time appropriately in its end-user interface.

Special Considerations

Releasing the observer object without invoking `removeTimeObserver:` (page 234) will result in undefined behavior.

Availability

Available in iOS 4.0 and later.

See Also

- `addBoundaryTimeObserverForTimes:queue:usingBlock:` (page 230)
- `removeTimeObserver:` (page 234)

`@property currentTime` (page 232)

Declared In

`AVPlayer.h`

currentTime

Returns the current time of the current item.

```
- (CMTime)currentTime
```

Return Value
The current time of the current item.

Discussion
This property is not key-value observable; use `addPeriodicTimeObserverForInterval:queue:usingBlock:` (page 231) or `addBoundaryTimeObserverForTimes:queue:usingBlock:` (page 230) instead.

Availability
Available in iOS 4.0 and later.

See Also
- `addPeriodicTimeObserverForInterval:queue:usingBlock:` (page 231)
- `addBoundaryTimeObserverForTimes:queue:usingBlock:` (page 230)

Declared In
`AVPlayer.h`

initWithPlayerItem:

Initializes a new player to play a given single audiovisual item.

```
- (id)initWithPlayerItem:(AVPlayerItem *)item
```

Parameters
item
A player item.

Return Value
The receiver, initialized to play *item*.

Discussion
You can use this method to play items for which you have an existing `AVAsset` object (see `initWithAsset:` (page 247) in `AVPlayerItem`).

Availability
Available in iOS 4.0 and later.

See Also
`+ playerWithPlayerItem:` (page 229)

Declared In
`AVPlayer.h`

initWithURL:

Initializes a new player to play a single audiovisual resource referenced by a given URL.

```
- (id)initWithURL:(NSURL *)URL
```

Parameters

URL

An URL that identifies an audiovisual resource.

Return Value

The receiver, initialized to play the audiovisual resource specified by *URL*.

Discussion

This method implicitly creates an `AVPlayerItem` object. You can get the player item using `currentItem` (page 227).

Availability

Available in iOS 4.0 and later.

See Also

`+ playerWithURL:` (page 229)
`@property currentItem` (page 227)

Declared In

`AVPlayer.h`

pause

Pauses playback.

```
- (void)pause
```

Discussion

This is the same as setting `rate` to `0.0`.

Availability

Available in iOS 4.0 and later.

See Also

`@property rate` (page 228)

Declared In

`AVPlayer.h`

play

Begins playback of the current item.

```
- (void)play
```

Discussion

This is the same as setting `rate` to `1.0`.

Availability

Available in iOS 4.0 and later.

See Also
`@property rate` (page 228)

Declared In
`AVPlayer.h`

removeTimeObserver:

Cancels a previously registered time observer.

```
- (void)removeTimeObserver:(id)observer
```

Parameters

observer

An object returned by a previous call to `addPeriodicTimeObserverForInterval:queue:usingBlock:` (page 231) or `addBoundaryTimeObserverForTimes:queue:usingBlock:` (page 230).

Discussion
Upon return, the caller is guaranteed that no new time observer blocks will begin executing. Depending on the calling thread and the queue used to add the time observer, an in-flight block may continue to execute after this method returns. You can guarantee synchronous time observer removal by enqueuing the call to `removeTimeObserver` on that queue. Alternatively, call `dispatch_sync(queue, ^{})` after `removeTimeObserver` to wait for any in-flight blocks to finish executing.

You should use this method to explicitly cancel each time observer added using -`addPeriodicTimeObserverForInterval:queue:usingBlock:` (page 231) and `addBoundaryTimeObserverForTimes:queue:usingBlock:` (page 230).

Availability
Available in iOS 4.0 and later.

Declared In
`AVPlayer.h`

replaceCurrentItemWithPlayerItem:

Replaces the player item with a new player item.

```
- (void)replaceCurrentItemWithPlayerItem:(AVPlayerItem *)item
```

Parameters

item

A player item.

Discussion
You use this method with players created without queues. If the player was not initialized with a single item and no queue, the method throws an exception.

The item replacement occurs asynchronously; observe the `currentItem` (page 227) property to find out when the replacement will/did occur.

Availability
Available in iOS 4.0 and later.

Declared In
AVPlayer.h

seekToTime:

Moves the playback cursor to a given time.

```
- (void)seekToTime:(CMTime)time
```

Parameters

time

The time to which to move the playback cursor.

Discussion
The time seeked to may differ from the specified time for efficiency. For sample accurate seeking see `seekToTime:toleranceBefore:toleranceAfter:` (page 235).

Availability
Available in iOS 4.0 and later.

Declared In
AVPlayer.h

seekToTime:toleranceBefore:toleranceAfter:

Moves the playback cursor within a specified time bound.

```
- (void)seekToTime:(CMTime)time toleranceBefore:(CMTime)toleranceBefore
    toleranceAfter:(CMTime)toleranceAfter
```

Parameters

time

The time to which you would like to move the playback cursor.

toleranceBefore

The tolerance allowed before *time*.

toleranceAfter

The tolerance allowed after *time*.

Discussion
The time seeked to will be within the range `[time-beforeTolerance, time+afterTolerance]`, and may differ from the specified time for efficiency. If you pass `kCMTimeZero` for both *toleranceBefore* and *toleranceAfter* (to request sample accurate seeking), you may incur additional decoding delay.

Passing `kCMTimePositiveInfinity` for both *toleranceBefore* and *toleranceAfter* is the same as messaging `seekToTime:` (page 235) directly.

Availability
Available in iOS 4.0 and later.

Declared In
AVPlayer.h

Constants

AVPlayerStatus

Possible values of the `status` (page 228) property, to indicate whether it can successfully play items.

```
enum {
    AVPlayerStatusUnknown,
    AVPlayerStatusReadyToPlay,
    AVPlayerStatusFailed
};
typedef NSInteger AVPlayerStatus;
```

Constants

`AVPlayerStatusUnknown`

Indicates that the status of the player is not yet known because it has not tried to load new media resources for playback.

Available in iOS 4.0 and later.

Declared in `AVPlayer.h`.

`AVPlayerStatusReadyToPlay`

Indicates that the player is ready to play `AVPlayerItem` instances.

Available in iOS 4.0 and later.

Declared in `AVPlayer.h`.

`AVPlayerStatusFailed`

Indicates that the player can no longer play `AVPlayerItem` instances because of an error.

The error is described by the value of the player's `error` (page 228) property.

Available in iOS 4.0 and later.

Declared in `AVPlayer.h`.

AVPlayerActionAtItemEnd

You use these constants with `actionAtItemEnd` (page 227) to indicate the action a player should take when it finishes playing.

```
enum
{
    AVPlayerActionAtItemEndPause,
    AVPlayerActionAtItemEndNone
};
typedef NSInteger AVPlayerActionAtItemEnd;
```

Constants

`AVPlayerActionAtItemEndPause`

Indicates that the player should pause playing.

Available in iOS 4.0 and later.

Declared in `AVPlayer.h`.

`AVPlayerActionAtItemEndNone`

Indicates that the player should do nothing.

Available in iOS 4.0 and later.

Declared in `AVPlayer.h`.

AVPlayerItem Class Reference

Inherits from	NSObject
Conforms to	NSCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/AVFoundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	AVPlayerItem.h

Overview

An `AVPlayerItem` represents the presentation state of an asset that's played by an `AVPlayer` object, and lets you observe that state.

A object carries a reference to an `AVAsset` object and presentation settings for that asset, including track enabled state. If you need to inspect the media assets themselves, you should message the `AVAsset` object itself.

You can initialize a player item using an URL (`playerItemWithURL:` (page 246) and `initWithURL:` (page 248)); the resource types referenced by the URL may include, but aren't necessarily limited to, those with the following corresponding UTIs:

`kUTTypeQuickTimeMovie`, (.mov, .qt)
`kUTTypeMPEG4` (.mp4)
`@"public.3gpp"` (.3gp, .3gpp)
`kUTTypeMPEG4Audio` (.m4a)
`@"com.apple.coreaudio-format"` (.caf)
`@"com.microsoft.waveform-audio"` (.wav)
`@"public.aiff-audio"` (.aif)
`@"public.aifc-audio"` (also .aif)
`@"org.3gpp.adaptive-multi-rate-audio"` (.amr)

If you want to play an asset more than once within a sequence of items, you must create independent instances of `AVPlayerItem` for each placement in the player's queue.

Tasks

Creating a Player Item

- `initWithURL:` (page 248)
  Prepares a player item with a given URL.
+ `playerItemWithURL:` (page 246)
  Returns a new player item, prepared to use a given URL.
- `initWithAsset:` (page 247)
  Initializes a new player item for a given asset.
+ `playerItemWithAsset:` (page 246)
  Returns a new player item for a given asset.

Getting Information About an Item

`asset` (page 241) *property*
The underlying asset provided during initialization. (read-only)

`tracks` (page 245) *property*
An array of `AVPlayerItemTrack` objects. (read-only)

`status` (page 245) *property*
The status of the player item. (read-only)

`loadedTimeRanges` (page 243) *property*
The time ranges of the item that have been loaded. (read-only)

`presentationSize` (page 244) *property*
The size at which the visual portion of the item is presented by the player. (read-only)

`timedMetadata` (page 245) *property*
The timed metadata played most recently by the media stream. (read-only)

`seekableTimeRanges` (page 244) *property*
(read-only)

`error` (page 242) *property*
If the receiver's status is `AVPlayerItemStatusFailed` (page 251), this describes the error that caused the failure. (read-only)

Moving the Playhead

- `stepByCount:` (page 250)
  Moves the player's current item's current time forward or backward by a specified number of steps.
- `seekToTime:` (page 249)
  Moves the playback cursor to a given time.
- `seekToTime:toleranceBefore:toleranceAfter:` (page 249)
  Moves the playback cursor within a specified time bound.

- `seekToDate:` (page 248)
  Moves the playback cursor to a given date.

Information About Playback

`playbackLikelyToKeepUp` (page 243) *property*
Indicates whether the item will likely play through without stalling (read-only)

`playbackBufferEmpty` (page 243) *property*
Indicates whether playback has consumed all buffered media and that playback will stall or end. (read-only)

`playbackBufferFull` (page 243) *property*
Indicates whether the internal media buffer is full and that further I/O is suspended. (read-only)

Timing Information

- `currentTime` (page 247)
  Returns the current time of the item.

`forwardPlaybackEndTime` (page 242) *property*
The time at which forward playback ends.

`reversePlaybackEndTime` (page 244) *property*
The time at which reverse playback ends.

Settings

`audioMix` (page 242) *property*
The audio mix parameters to be applied during playback.

`videoComposition` (page 245) *property*
The video composition settings to be applied during playback.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C Programming Language*.

asset

The underlying asset provided during initialization. (read-only)

```
@property(nonatomic, readonly) AVAsset *asset
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVPlayerItem.h`

audioMix

The audio mix parameters to be applied during playback.

```
@property(nonatomic, copy) AVAudioMix *audioMix
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVPlayerItem.h`

error

If the receiver's status is `AVPlayerItemStatusFailed` (page 251), this describes the error that caused the failure. (read-only)

```
@property(nonatomic, readonly) NSError *error
```

Discussion
The value of this property is an error that describes what caused the receiver to no longer be able to be played.

If the receiver's status is not `AVPlayerItemStatusFailed` (page 251), the value of this property is `nil`.

Availability
Available in iOS 4.0 and later.

Declared In
`AVPlayerItem.h`

forwardPlaybackEndTime

The time at which forward playback ends.

```
@property(nonatomic) CMTime forwardPlaybackEndTime
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVPlayerItem.h`

loadedTimeRanges

The time ranges of the item that have been loaded. (read-only)

```
@property(nonatomic, readonly) NSArray *loadedTimeRanges
```

Discussion
The array contains `NSValue` objects containing a `CMTimeRange` value.

Availability
Available in iOS 4.0 and later.

Declared In
`AVPlayerItem.h`

playbackBufferEmpty

Indicates whether playback has consumed all buffered media and that playback will stall or end. (read-only)

```
@property(nonatomic, readonly, getter=isPlaybackBufferEmpty) BOOL playbackBufferEmpty
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVPlayerItem.h`

playbackBufferFull

Indicates whether the internal media buffer is full and that further I/O is suspended. (read-only)

```
@property(nonatomic, readonly, getter=isPlaybackBufferFull) BOOL playbackBufferFull
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVPlayerItem.h`

playbackLikelyToKeepUp

Indicates whether the item will likely play through without stalling (read-only)

```
@property(nonatomic, readonly, getter=isPlaybackLikelyToKeepUp) BOOL
    playbackLikelyToKeepUp
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVPlayerItem.h`

presentationSize

The size at which the visual portion of the item is presented by the player. (read-only)

```
@property (nonatomic, readonly) CGSize presentationSize;
```

Discussion
You can scale the presentation size to fit within the bounds of a player layer using its `videoGravity` property.

Availability
Available in iOS 4.0 and later.

Declared In
`AVPlayerItem.h`

reversePlaybackEndTime

The time at which reverse playback ends.

```
@property(nonatomic) CMTime reversePlaybackEndTime
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVPlayerItem.h`

seekableTimeRanges

(read-only)

```
@property(nonatomic, readonly) NSArray *seekableTimeRanges
```

Discussion
The array contains `NSValue` objects containing a `CMTimeRange` value.

Availability
Available in iOS 4.0 and later.

Declared In
`AVPlayerItem.h`

status

The status of the player item. (read-only)

```
@property(nonatomic, readonly) AVPlayerItemStatus status
```

Discussion
For example, whether the item is playable. For possible values, see "AVPlayerItemStatus" (page 250).

Availability
Available in iOS 4.0 and later.

Declared In
`AVPlayerItem.h`

timedMetadata

The timed metadata played most recently by the media stream. (read-only)

```
@property(nonatomic, readonly) NSArray *timedMetadata
```

Discussion
The array contains instances of `AVMetadataItem`.

Availability
Available in iOS 4.0 and later.

Declared In
`AVPlayerItem.h`

tracks

An array of `AVPlayerItemTrack` objects. (read-only)

```
@property(nonatomic, readonly) NSArray *tracks
```

Discussion
This property can change dynamically during playback. You can observe it using key-value observing.

Availability
Available in iOS 4.0 and later.

Declared In
`AVPlayerItem.h`

videoComposition

The video composition settings to be applied during playback.

```
@property(nonatomic, copy) AVVideoComposition *videoComposition
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
AVPlayerItem.h

Class Methods

playerItemWithAsset:

Returns a new player item for a given asset.

```
+ (AVPlayerItem *)playerItemWithAsset:(AVAsset *)asset
```

Parameters

asset
An asset to play.

Return Value
A new player item, initialized to play *asset*.

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
AVPlayerItem.h

playerItemWithURL:

Returns a new player item, prepared to use a given URL.

```
+ (AVPlayerItem *)playerItemWithURL:(NSURL *)URL
```

Parameters

URL
An URL.

Return Value
A new player item, prepared to use *URL*.

Special Considerations

This method immediately returns the item, but with the status `AVPlayerItemStatusUnknown` (page 250).

If the URL contains valid data that can be used by the player item, the status later changes to `AVPlayerItemStatusReadyToPlay` (page 251).

If the URL contains no valid data or otherwise can't be used by the player item, the status later changes to `AVPlayerItemStatusFailed` (page 251).

Availability
Available in iOS 4.0 and later.

See Also
`@property status` (page 245)

Declared In
`AVPlayerItem.h`

Instance Methods

currentTime

Returns the current time of the item.

```
- (CMTime)currentTime
```

Return Value
The current time of the item.

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVPlayerItem.h`

initWithAsset:

Initializes a new player item for a given asset.

```
- (id)initWithAsset:(AVAsset *)asset
```

Parameters

asset
An asset to play.

Return Value
The receiver, initialized to play *asset*.

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVPlayerItem.h`

initWithURL:

Prepares a player item with a given URL.

```
- (id)initWithURL:(NSURL *)URL
```

Parameters

URL

An URL.

Return Value

The receiver, prepared to use *URL*.

Special Considerations

This method immediately returns the item, but with the status `AVPlayerItemStatusUnknown` (page 250).

If the URL contains valid data that can be used by the player item, the status later changes to `AVPlayerItemStatusReadyToPlay` (page 251).

If the URL contains no valid data or otherwise can't be used by the player item, the status later changes to `AVPlayerItemStatusFailed` (page 251).

Availability

Available in iOS 4.0 and later.

See Also

`@property status` (page 245)

Declared In

`AVPlayerItem.h`

seekToDate:

Moves the playback cursor to a given date.

```
- (BOOL)seekToDate:(NSDate *)date
```

Parameters

date

The date to which to move the playback cursor.

Return Value

`YES` if the playhead was moved to *date*, otherwise `NO`.

Discussion

For playback content that is associated with a range of dates, this method moves the playhead to point within that range. This method will fail (return `NO`) if *date* is outside the range or if the content is not associated with a range of dates.

Availability

Available in iOS 4.0 and later.

See Also

- `- seekToTime:` (page 249)
- `- seekToDate:` (page 248)

Declared In
AVPlayerItem.h

seekToTime:

Moves the playback cursor to a given time.

```
- (void)seekToTime:(CMTime)time
```

Parameters

time
The time to which to move the playback cursor.

Discussion
The time seeked to may differ from the specified time for efficiency. For sample accurate seeking see `seekToTime:toleranceBefore:toleranceAfter:` (page 249).

Availability
Available in iOS 4.0 and later.

See Also
- `seekToTime:toleranceBefore:toleranceAfter:` (page 249)
- `seekToDate:` (page 248)

Declared In
AVPlayerItem.h

seekToTime:toleranceBefore:toleranceAfter:

Moves the playback cursor within a specified time bound.

```
- (void)seekToTime:(CMTime)time toleranceBefore:(CMTime)toleranceBefore
    toleranceAfter:(CMTime)toleranceAfter
```

Parameters

time
The time to which you would like to move the playback cursor.

toleranceBefore
The tolerance allowed before *time*.

toleranceAfter
The tolerance allowed after *time*.

Discussion
The time seeked to will be within the range `[time-beforeTolerance, time+afterTolerance]`, and may differ from the specified time for efficiency. If you pass `kCMTimeZero` for both *toleranceBefore* and *toleranceAfter* (to request sample accurate seeking), you may incur additional decoding delay.

Passing `kCMTimePositiveInfinity` for both *toleranceBefore* and *toleranceAfter* is the same as messaging `seekToTime:` (page 249) directly.

Availability
Available in iOS 4.0 and later.

See Also
- `seekToTime:` (page 249)
- `seekToDate:` (page 248)

Declared In
`AVPlayerItem.h`

stepByCount:

Moves the player's current item's current time forward or backward by a specified number of steps.

```
- (void)stepByCount:(NSInteger)stepCount
```

Parameters
stepCount
The number of steps by which to move.

A positive number steps forward, a negative number steps backward.

Discussion
The size of each step depends on the receiver's enabled `AVPlayerItemTrack` objects (see `tracks` (page 245)).

Availability
Available in iOS 4.0 and later.

Declared In
`AVPlayerItem.h`

Constants

AVPlayerItemStatus

Constants that represent the status of an item

```
enum {
    AVPlayerItemStatusUnknown,
    AVPlayerItemStatusReadyToPlay,
    AVPlayerItemStatusFailed
};
typedef NSInteger AVPlayerItemStatus;
```

Constants
`AVPlayerItemStatusUnknown`
The item's status is unknown.

Available in iOS 4.0 and later.

Declared in `AVPlayerItem.h`.

`AVPlayerItemStatusReadyToPlay`
The item is ready to play.

Available in iOS 4.0 and later.

Declared in `AVPlayerItem.h`.

`AVPlayerItemStatusFailed`
The item cannot be played.

Available in iOS 4.0 and later.

Declared in `AVPlayerItem.h`.

Notifications

AVPlayerItemDidPlayToEndTimeNotification

Posted when the item has played to its end time.

The notification's object is the item that finished playing.

Important: This notification may be posted on a different thread than the one on which the observer was registered.

Availability
Available in iOS 4.0 and later.

Declared In
`AVPlayerItem.h`

AVPlayerItemTrack Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/AVFoundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	AVPlayerItemTrack.h

Overview

You use an `AVPlayerItemTrack` object to modify the presentation state of an asset track (`AVAssetTrack`) being presented by an `AVPlayer` object.

Tasks

Properties

`assetTrack` (page 253) *property*
The asset track for which the player item represents presentation state. (read-only)

`enabled` (page 254) *property*
Indicates whether the track is enabled for presentation during playback.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C Programming Language*.

assetTrack

The asset track for which the player item represents presentation state. (read-only)

```
@property(nonatomic, readonly) AVAssetTrack *assetTrack
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVPlayerItemTrack.h`

enabled

Indicates whether the track is enabled for presentation during playback.

```
@property(nonatomic, assign, getter=isEnabled) BOOL enabled
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVPlayerItemTrack.h`

AVPlayerLayer Class Reference

Inherits from	CALayer : NSObject
Conforms to	NSCoding (CALayer) CAMediaTiming (CALayer) NSObject (NSObject)
Framework	/System/Library/Frameworks/AVFoundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	AVPlayerLayer.h

Overview

`AVPlayerLayer` is a subclass of `CALayer` to which an `AVPlayer` object can direct its visual output.

You can create arbitrary numbers of player layers with the same `AVPlayer` object. You can create a layer as illustrated in the following code fragment:

```
AVPlayer *player = <#A configured AVPlayer object#>;

CALayer *superlayer = <#Get a CALayer#>;
AVPlayerLayer *playerLayer = [AVPlayerLayer playerLayerWithPlayer:player];
[superlayer addSublayer:playerLayer];
```

The value for the `contents` key of a player layer is opaque and effectively read-only.

During playback, `AVPlayer` may compensate for temporal drift between its visual output and its audible output to one or more independently-clocked audio output devices by adjusting the timing of its associated player layers. The effects of these adjustments are usually very small; however, clients that wish to remain entirely unaffected by such adjustments may wish to place other layers for which timing is important into independently timed subtrees of their layer trees.

Tasks

Miscellaneous

`player` (page 256) *property*
The player for which the player layer displays visual output.

`+ playerLayerWithPlayer:` (page 257)
Returns a player layer to display the visual output of a specified player.

`readyForDisplay` (page 256) *property*
Indicates whether the player is ready to be displayed. (read-only)

`videoGravity` (page 256) *property*
Specifies how the video is displayed within a player layer's bounds.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C Programming Language*.

player

The player for which the player layer displays visual output.

```
@property(nonatomic, retain) AVPlayer *player
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVPlayerLayer.h`

readyForDisplay

Indicates whether the player is ready to be displayed. (read-only)

```
@property(nonatomic, readonly, getter=isReadyForDisplay) BOOL readyForDisplay
```

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVPlayerLayer.h`

videoGravity

Specifies how the video is displayed within a player layer's bounds.

```
@property(copy) NSString *videoGravity
```

Discussion
The default is `AVPlayerLayerVideoGravityResizeAspect` (page ?).

This property is animatable.

Availability
Available in iOS 4.0 and later.

Declared In
`AVPlayerLayer.h`

Class Methods

playerLayerWithPlayer:

Returns a player layer to display the visual output of a specified player.

```
+ (AVPlayerLayer *)playerLayerWithPlayer:(AVPlayer *)player
```

Parameters

player
The player for which the player layer displays visual output.

Return Value
A player layer configured to display the visual output of *player*.

Discussion

Availability
Available in iOS 4.0 and later.

Declared In
`AVPlayerLayer.h`

AVSynchronizedLayer Class Reference

Inherits from	CALayer : NSObject
Conforms to	NSCoding (CALayer) CAMediaTiming (CALayer) NSObject (NSObject)
Framework	/System/Library/Frameworks/AVFoundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	AVSynchronizedLayer.h

Overview

`AVSynchronizedLayer` a subclass of `CALayer` with layer timing that synchronizes with a specific `AVPlayerItem`.

You can create an arbitrary number of synchronized layers from the same `AVPlayerItem` object.

An synchronized layers is similar to a `CATransformLayer` object in that it doesn't display anything itself but only confers state upon its layer subtree. `AVSynchronizedLayer` confers is timing state, synchronizing the timing of layers in its subtree with that of a player item.

You might use a layer as shown in the following example:

```
AVPlayerItem *playerItem = <#Get a player item#>;
CALayer *superLayer =  <#Get a layer#>;
// Set up a synchronized layer to sync the layer timing of its subtree
// with the playback of the playerItem
AVSynchronizedLayer *syncLayer = [AVSynchronizedLayer
synchronizedLayerWithPlayerItem:playerItem];
[syncLayer addSublayer:<#Another layer#>];    // These sublayers will be
synchronized
[superLayer addSublayer:syncLayer];
```

Tasks

Creating a Synchronized Layer

`+ synchronizedLayerWithPlayerItem:` (page 260)

Returns a new synchronized layer with timing synchronized with a given player item.

Managing the Player Item

`playerItem` (page 260) *property*
The player item to which the timing of the layer is synchronized.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C Programming Language*.

playerItem

The player item to which the timing of the layer is synchronized.

```
@property(nonatomic, retain) AVPlayerItem *playerItem
```

Availability
Available in iOS 4.0 and later.

Declared In
`AVSynchronizedLayer.h`

Class Methods

synchronizedLayerWithPlayerItem:

Returns a new synchronized layer with timing synchronized with a given player item.

```
+ (AVSynchronizedLayer *)synchronizedLayerWithPlayerItem:(AVPlayerItem *)playerItem
```

Parameters
playerItem
A player item.

Return Value
A new synchronized layer with timing synchronized with *playerItem*.

Availability
Available in iOS 4.0 and later.

Declared In
`AVSynchronizedLayer.h`

AVURLAsset Class Reference

Inherits from	AVAsset : NSObject
Conforms to	NSCopying (AVAsset) AVAsynchronousKeyValueLoading (AVAsset) NSObject (NSObject)
Framework	/System/Library/Frameworks/AVFoundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	AVAsset.h

Overview

`AVURLAsset` is a concrete subclass of `AVAsset` that you use to initialize an asset from an URL.

Tasks

Creating an URL Asset

`- initWithURL:options:` (page 263)
Initializes an asset for inspection of a resource referenced by a given URL.

`+ URLAssetWithURL:options:` (page 262)
Returns an asset for inspection of a resource referenced by a given URL.

Accessing the URL

`URL` (page 262) *property*
The URL with which the asset was initialized. (read-only)

Finding Compatible Tracks

`- compatibleTrackForCompositionTrack:` (page 263)
Returns an asset track from which any time range can be inserted into a given composition track.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C Programming Language*.

URL

The URL with which the asset was initialized. (read-only)

```
@property(nonatomic, readonly, copy) NSURL *URL
```

Availability
Available in iOS 4.0 and later.

See Also
- `initWithURL:options:` (page 263)
+ `URLAssetWithURL:options:` (page 262)

Declared In
`AVAsset.h`

Class Methods

URLAssetWithURL:options:

Returns an asset for inspection of a resource referenced by a given URL.

```
+ (AVURLAsset *)URLAssetWithURL:(NSURL *)URL options:(NSDictionary *)options
```

Parameters

URL

An URL that references the container file to be represented by the asset.

options

A dictionary that contains options for the initialization of the asset.

For possible keys and values, see "Initialization Options" (page 264).

Return Value
An asset initialized for inspection of a resource referenced by *URL*.

Availability
Available in iOS 4.0 and later.

See Also
- `initWithURL:options:` (page 263)
`@property URL` (page 262)

Declared In
`AVAsset.h`

Instance Methods

compatibleTrackForCompositionTrack:

Returns an asset track from which any time range can be inserted into a given composition track.

```
- (AVAssetTrack *)compatibleTrackForCompositionTrack:(AVCompositionTrack
    *)compositionTrack
```

Parameters

compositionTrack

The composition track for which a compatible `AVAssetTrack` object is requested.

Return Value

An asset track managed by the receiver from which any time range can be inserted into a given composition track.

Discussion

You insert the track into using `insertTimeRange:ofTrack:atTime:error:` (page 203) (`AVMutableCompositionTrack`). This method is the logical complement of `mutableTrackCompatibleWithTrack:` (page 195).

Availability

Available in iOS 4.0 and later.

Declared In

`AVAsset.h`

initWithURL:options:

Initializes an asset for inspection of a resource referenced by a given URL.

```
- (id)initWithURL:(NSURL *)URL options:(NSDictionary *)options
```

Parameters

URL

An URL that references the container file to be represented by the asset.

options

A dictionary that contains options for the initialization of the asset.

For possible keys and values, see "Initialization Options" (page 264).

Return Value

An asset initialized for inspection of a resource referenced by *URL*.

Availability

Available in iOS 4.0 and later.

See Also

`+ URLAssetWithURL:options:` (page 262)

`@property URL` (page 262)

Declared In
`AVAsset.h`

Constants

Initialization Options

Keys for options dictionary for use with `initWithURL:options:` (page 263) and `URLAssetWithURL:options:` (page 262).

```
NSString *const AVURLAssetPreferPreciseDurationAndTimingKey;
```

Constants

`AVURLAssetPreferPreciseDurationAndTimingKey`

The corresponding value is a boolean, contained in an `NSValue` object, that indicates whether the asset should be prepared to indicate a precise duration and provide precise random access by time.

`YES` indicates that longer loading times are acceptable in cases in which precise timing is required. Such precision, however, may require additional parsing of the resource in advance of operations that make use of any portion of it, depending on the specifics of its container format.

Many container formats provide sufficient summary information for precise timing and do not require additional parsing to prepare for it; QuickTime movie files and MPEG-4 files are examples of such formats. Other formats do not provide sufficient summary information, and precise random access for them is possible only after a preliminary examination of a file's contents.

If you only intend that the asset be played, the default value of `NO` will suffice (because `AVPlayer` supports approximate random access by time when full precision isn't available). If you intend to insert the asset into an `AVMutableComposition` object, precise random access is typically desirable, and the value of `YES` is recommended.

Available in iOS 4.0 and later.

Declared in `AVAsset.h`.

AVVideoComposition Class Reference

Inherits from	NSObject
Conforms to	NSCopying NSMutableCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/AVFoundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	AVVideoComposition.h

Overview

An `AVVideoComposition` object represents an immutable video composition.

The AVFoundation framework also provides a mutable subclass, `AVMutableVideoComposition`, that you can use to create new videos.

Tasks

Properties

`frameDuration` (page 266) *property*
The interval for which the video composition should render composed video frames. (read-only)

`renderSize` (page 267) *property*
The size at which the video composition should render. (read-only)

`instructions` (page 266) *property*
The video composition instructions. (read-only)

`animationTool` (page 266) *property*
A video composition tool to use with Core Animation in offline rendering. (read-only)

`renderScale` (page 267) *property*
The scale at which the video composition should render.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C Programming Language*.

animationTool

A video composition tool to use with Core Animation in offline rendering. (read-only)

```
@property(nonatomic, readonly, retain) AVVideoCompositionCoreAnimationTool
    *animationTool
```

Discussion
This attribute may be `nil`.

You set an animation tool if you are using the composition in conjunction with `AVAssetExportSession` for offline rendering, rather than with `AVPlayer`.

Availability
Available in iOS 4.0 and later.

Declared In
`AVVideoComposition.h`

frameDuration

The interval for which the video composition should render composed video frames. (read-only)

```
@property(nonatomic, readonly) CMTime frameDuration
```

Discussion
This property only applies when the composition is enabled.

Availability
Available in iOS 4.0 and later.

Declared In
`AVVideoComposition.h`

instructions

The video composition instructions. (read-only)

```
@property(nonatomic, readonly, copy) NSArray *instructions
```

Discussion
The array contains of instances of `AVVideoCompositionInstruction`.

For the first instruction in the array, `timeRange.start` must be less than or equal to the earliest time for which playback or other processing will be attempted (typically `kCMTimeZero`). For subsequent instructions, `timeRange.start` must be equal to the prior instruction's end time. The end time of the last instruction must be greater than or equal to the latest time for which playback or other processing will be attempted (typically be the duration of the asset with which the instance of `AVVideoComposition` is associated).

Availability
Available in iOS 4.0 and later.

Declared In
`AVVideoComposition.h`

renderScale

The scale at which the video composition should render.

```
@property (nonatomic, readonly) float renderScale
```

Discussion
This value must be 1.0 unless the composition is set on an `AVPlayerItem`.

Availability
Available in iOS 4.0 and later.

Declared In
`AVVideoComposition.h`

renderSize

The size at which the video composition should render. (read-only)

```
@property(nonatomic, readonly) CGSize renderSize
```

Discussion
This property only applies when the composition is enabled.

Availability
Available in iOS 4.0 and later.

Declared In
`AVVideoComposition.h`

AVVideoCompositionInstruction Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSCopying NSMutableCopying NSObject (NSObject)
Framework	/System/Library/Frameworks/AVFoundation.framework
Availability	Available in iOS 4.0 and later.
Declared in	AVVideoComposition.h

Overview

An `AVVideoCompositionInstruction` object represents an operation to be performed by a compositor.

An `AVVideoComposition` object maintains an array of `instructions` to perform its composition.

Tasks

Properties

`backgroundColor` (page 270) *property*
The background color of the composition.

`layerInstructions` (page 270) *property*
An array of instances of `AVVideoCompositionLayerInstruction` that specify how video frames from source tracks should be layered and composed. (read-only)

`timeRange` (page 270) *property*
The time range during which the instruction is effective. (read-only)

Properties

For more about Objective-C properties, see “Properties” in *The Objective-C Programming Language*.

backgroundColor

The background color of the composition.

```
@property(nonatomic, retain) CGColorRef backgroundColor
```

Discussion
Only solid BGRA colors are supported; patterns and other color refs that are not supported are ignored. If the rendered pixel buffer does not have alpha, the alpha value of the background color is ignored.

If the background color is not specified, the video compositor will use a default background color of opaque black.

Availability
Available in iOS 4.0 and later.

Declared In
`AVVideoComposition.h`

layerInstructions

An array of instances of `AVVideoCompositionLayerInstruction` that specify how video frames from source tracks should be layered and composed. (read-only)

```
@property(nonatomic, readonly, copy) NSArray *layerInstructions
```

Discussion
Tracks are layered in the composition according to the top-to-bottom order of the `layerInstructions` array; the track with trackID of the first instruction in the array will be layered on top, with the track with the trackID of the second instruction immediately underneath, and so on.

If this key is `nil`, the output will be a fill of the background color.

Availability
Available in iOS 4.0 and later.

See Also
`@property backgroundColor` (page 270)

Declared In
`AVVideoComposition.h`

timeRange

The time range during which the instruction is effective. (read-only)

```
@property(nonatomic, readonly) CMTimeRange timeRange
```

Discussion
If the time range is invalid, the video compositor will ignore it.

enablePostProcessing

Indicates whether post-processing should be allowed for the duration of the instruction. (read-only)

```
@property(nonatomic, readonly) BOOL enablePostProcessing
```

Discussion
`NO` indicates that post-processing should be skipped for the duration of this instruction.

The value is `YES` by default.

Availability
Available in iOS 4.0 and later.

Declared In
`AVVideoComposition.h`

Availability
Available in iOS 4.0 and later.

Declared In
`AVVideoComposition.h`

NSCoder AV Foundation Additions Reference

Inherits from	NSObject
Framework	/System/Library/Frameworks/AVFoundation.framework
Declared in	AVTime.h
Companion guide	Archives and Serializations Programming Guide

Overview

The AV Foundation framework adds methods to the `NSCoder` class to make it easier to create archives including Core Media time structures, and extract Core Media time structure from archives.

Tasks

Encoding Core Media Time Structures

- `encodeCMTime:forKey:` (page 275)
  Encodes a given `CMTime` structure and associates it with a specified key.
- `encodeCMTimeRange:forKey:` (page 276)
  Encodes a given `CMTimeRange` structure and associates it with a specified key.
- `encodeCMTimeMapping:forKey:` (page 275)
  Encodes a given `CMTimeMapping` structure and associates it with a specified key.

Decoding Core Media Time Structures

- `decodeCMTimeForKey:` (page 274)
  Returns the `CMTime` structure associated with a given key.
- `decodeCMTimeRangeForKey:` (page 274)
  Returns the `CMTimeRange` structure associated with a given key.
- `decodeCMTimeMappingForKey:` (page 274)
  Returns the `CMTimeMapping` structure associated with a given key.

Instance Methods

decodeCMTimeForKey:

Returns the `CMTime` structure associated with a given key.

```
- (CMTime)decodeCMTimeForKey:(NSString *)key
```

Parameters

key

The key for a `CMTime` structure encoded in the receiver.

Return Value

The `CMTime` structure associated with *key* in the archive.

Availability

Available in iOS 4.0 and later.

See Also

`- encodeCMTime:forKey:` (page 275)

Declared In

`AVTime.h`

decodeCMTimeMappingForKey:

Returns the `CMTimeMapping` structure associated with a given key.

```
- (CMTimeMapping)decodeCMTimeMappingForKey:(NSString *)key
```

Parameters

key

The key for a `CMTimeMapping` structure encoded in the receiver.

Return Value

The `CMTimeMapping` structure associated with *key* in the archive.

Availability

Available in iOS 4.0 and later.

See Also

`- encodeCMTimeMapping:forKey:` (page 275)

Declared In

`AVTime.h`

decodeCMTimeRangeForKey:

Returns the `CMTimeRange` structure associated with a given key.

```
- (CMTimeRange)decodeCMTimeRangeForKey:(NSString *)key
```

Parameters

key

The key for a `CMTimeRange` structure encoded in the receiver.

Return Value

The `CMTimeRange` structure associated with *key* in the archive.

Availability

Available in iOS 4.0 and later.

See Also

`- encodeCMTimeRange:forKey:` (page 276)

Declared In

`AVTime.h`

encodeCMTime:forKey:

Encodes a given `CMTime` structure and associates it with a specified key.

```
- (void)encodeCMTime:(CMTime)time forKey:(NSString *)key
```

Parameters

time

A `CMTime` structure.

key

The key with which to associate *time* in the archive.

Availability

Available in iOS 4.0 and later.

See Also

`- decodeCMTimeRangeForKey:` (page 274)

Declared In

`AVTime.h`

encodeCMTimeMapping:forKey:

Encodes a given `CMTimeMapping` structure and associates it with a specified key.

```
- (void)encodeCMTimeMapping:(CMTimeMapping)timeMapping
    forKey:(NSString *)key
```

Parameters

timeMapping

A `CMTimeMapping` structure.

key

The key with which to associate *timeMapping* in the archive.

Availability

Available in iOS 4.0 and later.

See Also
- `decodeCMTimeMappingForKey:` (page 274)

Declared In
`AVTime.h`

encodeCMTimeRange:forKey:

Encodes a given `CMTimeRange` structure and associates it with a specified key.

```
- (void)encodeCMTimeRange:(CMTimeRange)timeRange forKey:(NSString *)key
```

Parameters

timeRange
A `CMTimeRange` structure.

key
The key with which to associate *timeRange* in the archive.

Availability
Available in iOS 4.0 and later.

See Also
- `decodeCMTimeRangeForKey:` (page 274)

Declared In
`AVTime.h`

NSValue AV Foundation Additions Reference

Inherits from	NSObject
Framework	/System/Library/Frameworks/AVFoundation.framework
Declared in	AVTime.h

Overview

The AVFoundation framework adds methods to the `NSValue` class to make it easier to create a value object with a Core Media time structure, and extract a Core Media time structure from a value object.

Tasks

Creating a Value Object

`+ valueWithCMTime:` (page 278)
Returns a value object that contains a given `CMTime` structure.

`+ valueWithCMTimeMapping:` (page 278)
Returns a value object that contains a given `CMTimeMapping` structure.

`+ valueWithCMTimeRange:` (page 278)
Returns a value object that contains a given `CMTimeRange` structure.

Retrieving Core Media Time Structures

`- CMTimeMappingValue` (page 279)
Returns a `CMTimeMapping` structure representation of the receiver.

`- CMTimeRangeValue` (page 279)
Returns a `CMTimeRange` structure representation of the receiver.

`- CMTimeValue` (page 280)
Returns a `CMTime` structure representation of the receiver.

Class Methods

valueWithCMTime:

Returns a value object that contains a given `CMTime` structure.

```
+ (NSValue *)valueWithCMTime:(CMTime)time
```

Parameters

time

A time.

Return Value

A value object initialized using *time*.

Availability

Available in iOS 4.0 and later.

See Also

- `CMTimeValue` (page 280)

Declared In

`AVTime.h`

valueWithCMTimeMapping:

Returns a value object that contains a given `CMTimeMapping` structure.

```
+ (NSValue *)valueWithCMTimeMapping:(CMTimeMapping)timeMapping
```

Parameters

timeMapping

A time mapping.

Return Value

A value object initialized using *timeMapping*.

Availability

Available in iOS 4.0 and later.

See Also

- `CMTimeMappingValue` (page 279)

Declared In

`AVTime.h`

valueWithCMTimeRange:

Returns a value object that contains a given `CMTimeRange` structure.

```
+ (NSValue *)valueWithCMTimeRange:(CMTimeRange)timeRange
```

Parameters

timeRange

A time range.

Return Value

A value object initialized using *timeRange*.

Availability

Available in iOS 4.0 and later.

See Also

- CMTimeRangeValue (page 279)

Declared In

AVTime.h

Instance Methods

CMTimeMappingValue

Returns a CMTimeMapping structure representation of the receiver.

```
- (CMTimeMapping)CMTimeMappingValue
```

Return Value

A CMTimeMapping structure representation of the receiver.

Availability

Available in iOS 4.0 and later.

See Also

+ valueWithCMTimeMapping: (page 278)

Declared In

AVTime.h

CMTimeRangeValue

Returns a CMTimeRange structure representation of the receiver.

```
- (CMTimeRange)CMTimeRangeValue
```

Return Value

A CMTimeRange structure representation of the receiver.

Availability

Available in iOS 4.0 and later.

See Also

+ valueWithCMTimeRange: (page 278)

Declared In
AVTime.h

CMTimeValue

Returns a CMTime structure representation of the receiver.

```
- (CMTime)CMTimeValue
```

Return Value
A CMTime structure representation of the receiver.

Availability
Available in iOS 4.0 and later.

See Also
+ valueWithCMTime: (page 278)

Declared In
AVTime.h

Protocols

AVAsynchronousKeyValueLoading Protocol Reference

Framework	/System/Library/Frameworks/AVFoundation.framework/
Availability	Available in iOS 4.0 and later.
Declared in	AVAsynchronousKeyValueLoading.h

Overview

The `AVAsynchronousKeyValueLoading` protocol defines methods that let you use an `AVAsset` or `AVAssetTrack` object without blocking a thread. Using methods in the protocol, you can find out the current status of a key (for example, whether the corresponding value has been loaded); and ask the object to load values asynchronously, informing you when the operation has completed.

Because of the nature of timed audiovisual media, successful initialization of an asset does not necessarily mean that all its data is immediately available. Instead, an asset will wait to load data until an operation is performed on it (for example, directly invoking any relevant`AVAsset` methods, playback via an `AVPlayerItem` object, export using `AVAssetExportSession`, reading using an instance of `AVAssetReader`, and so on). This means that although you can request the value of any key at any time, and its value will be returned synchronously, the calling thread may be blocked until the request can be satisfied. To avoid blocking, you can:

- First, determine whether the value for a given key (or given keys) is available, using `statusOfValueForKey:error:` (page 285).
- If the value has not (or values have not) been loaded yet, you can ask for them o be loaded and to be notified when their values become available using `loadValuesAsynchronouslyForKeys:completionHandler:` (page 284).

Even for use cases that may typically support ready access to some keys (such as for assets initialized with URLs for files in the local filesystem), slow I/O may require `AVAsset` to block before returning their values. Although blocking may be acceptable in cases in which you are preparing assets on background threads or in operation queues, in all cases in which blocking should be avoided you should use `loadValuesAsynchronouslyForKeys:completionHandler:` (page 284).

Tasks

Protocol Methods

- `loadValuesAsynchronouslyForKeys:completionHandler:` (page 284) *required method*
  Tells the asset to load the values of any of the specified keys that are not already loaded. (required)
- `statusOfValueForKey:error:` (page 285) *required method*
  Reports whether the value for a given key is immediately available without blocking. (required)

Instance Methods

loadValuesAsynchronouslyForKeys:completionHandler:

Tells the asset to load the values of any of the specified keys that are not already loaded. (required)

```
- (void)loadValuesAsynchronouslyForKeys:(NSArray *)keys completionHandler:(void
    (^)(void))handler
```

Parameters

keys
An array containing the required keys.
A key is an instance of `NSString`.

handler
The block to be invoked when loading succeeds, fails, or is cancelled.

Discussion
The completion handler will be invoked exactly once per invocation of this method:

- Synchronously if an I/O error or other format-related error occurs immediately.
- Asynchronously at a subsequent time if a loading error occurs at a later stage of processing, or if `cancelLoading` (page 23) is invoked on an `AVAsset` instance.

The completion states of the keys you specify in *keys* are not necessarily the same—some may be loaded, and others may have failed. You must check the status of each key individually.

If you want to receive error reporting for loading that's still pending, you can call this method at any time—even after an asset has begun to load data for operations in progress or already completed. If a fatal error has already occurred, the completion handler is invoked synchronously.

Availability
Available in iOS 4.0 and later.

See Also
- `statusOfValueForKey:error:` (page 285)

Declared In
`AVAsynchronousKeyValueLoading.h`

statusOfValueForKey:error:

Reports whether the value for a given key is immediately available without blocking. (required)

```
- (AVKeyValueStatus)statusOfValueForKey:(NSString *)key
    error:(NSError **)outError
```

Parameters

key

The key whose status you want.

key

If the status of the value for the *key* is `AVKeyValueStatusFailed` (page 286), upon return contains an `NSError` object that describes the failure that occurred.

Return Value

The current loading status of the value for *key*. For possible values, see "Protocol Methods" (page 284).

Discussion

You use this method to determine the availability of the value for a key. This method does not cause an asset to load the value of a key that's not yet available. To request values for keys that may not already be loaded without blocking, use `loadValuesAsynchronouslyForKeys:completionHandler:` (page 284) and wait for invocation of the completion handler to be informed of availability.

Availability

Available in iOS 4.0 and later.

See Also

`- loadValuesAsynchronouslyForKeys:completionHandler:` (page 284)

Declared In

`AVAsynchronousKeyValueLoading.h`

Constants

AVKeyValueStatus

A type to specify the load status of a given property.

```
typedef NSInteger AVKeyValueStatus;
```

Discussion

For possible values, see "Key Loading Status" (page 286).

Availability

Available in iOS 4.0 and later.

Declared In

`AVAsynchronousKeyValueLoading.h`

Key Loading Status

Constants to indicate the load status of a property.

```
enum {
    AVKeyValueStatusUnknown,
    AVKeyValueStatusLoading,
    AVKeyValueStatusLoaded,
    AVKeyValueStatusFailed,
    AVKeyValueStatusCancelled
};
```

Constants

`AVKeyValueStatusUnknown`
Indicates that the property status is unknown.
Available in iOS 4.0 and later.
Declared in `AVAsynchronousKeyValueLoading.h`.

`AVKeyValueStatusLoading`
Indicates that the property is not fully loaded.
Available in iOS 4.0 and later.
Declared in `AVAsynchronousKeyValueLoading.h`.

`AVKeyValueStatusLoaded`
Indicates that the property is ready for use.
Available in iOS 4.0 and later.
Declared in `AVAsynchronousKeyValueLoading.h`.

`AVKeyValueStatusFailed`
Indicates that the attempt to load the property failed.
Available in iOS 4.0 and later.
Declared in `AVAsynchronousKeyValueLoading.h`.

`AVKeyValueStatusCancelled`
Indicates that the attempt to load the property was cancelled.
Available in iOS 4.0 and later.
Declared in `AVAsynchronousKeyValueLoading.h`.

Discussion

See also `statusOfValueForKey:error:` (page 285).

AVAudioPlayerDelegate Protocol Reference

Conforms to	NSObject
Framework	/System/Library/Frameworks/AVFoundation.framework
Availability	Available in iOS 2.2 and later.
Declared in	AVAudioPlayer.h
Related sample code	AddMusic

Overview

The delegate of an `AVAudioPlayer` object must adopt the `AVAudioPlayerDelegate` protocol. All of the methods in this protocol are optional. They allow a delegate to respond to audio interruptions and audio decoding errors, and to the completion of a sound's playback.

Tasks

Responding to Sound Playback Completion

`- audioPlayerDidFinishPlaying:successfully:` (page 288)
Called when a sound has finished playing.

Responding to an Audio Decoding Error

`- audioPlayerDecodeErrorDidOccur:error:` (page 288)
Called when an audio player encounters a decoding error during playback.

Handling Audio Interruptions

`- audioPlayerBeginInterruption:` (page 288)
Called when an audio player is interrupted, such as by an incoming phone call.

`- audioPlayerEndInterruption:` (page 289)
Called after your audio session interruption ends.

- audioPlayerEndInterruption:withFlags: (page 289)
  Called after your audio session interruption ends, with flags indicating the state of the audio session.

Instance Methods

audioPlayerBeginInterruption:

Called when an audio player is interrupted, such as by an incoming phone call.

```
- (void)audioPlayerBeginInterruption:(AVAudioPlayer *)player
```

Parameters

player
  The audio player that has been interrupted.

Discussion
Upon interruption, your application's audio session is deactivated and the audio player pauses. You cannot use the audio player again until you receive a notification that the interruption has ended.

Availability
Available in iOS 2.2 and later.

See Also
- audioPlayerEndInterruption:withFlags: (page 289)

Declared In
AVAudioPlayer.h

audioPlayerDecodeErrorDidOccur:error:

Called when an audio player encounters a decoding error during playback.

```
- (void)audioPlayerDecodeErrorDidOccur:(AVAudioPlayer *)player error:(NSError
    *)error
```

Parameters

player
  The audio player that encountered the decoding error.

error
  The decoding error.

Availability
Available in iOS 2.2 and later.

Declared In
AVAudioPlayer.h

audioPlayerDidFinishPlaying:successfully:

Called when a sound has finished playing.

```
- (void)audioPlayerDidFinishPlaying:(AVAudioPlayer *)player successfully:(BOOL)flag
```

Parameters

player

The audio player that finished playing.

flag

YES on successful completion of playback; NO if playback stopped because the system could not decode the audio data.

Discussion

This method is not called upon an audio interruption. Rather, an audio player is paused upon interruption—the sound has not finished playing.

Availability

Available in iOS 2.2 and later.

Declared In

AVAudioPlayer.h

audioPlayerEndInterruption:

Called after your audio session interruption ends.

```
- (void)audioPlayerEndInterruption:(AVAudioPlayer *)player
```

Parameters

player

The audio player whose interruption has ended.

Discussion

If you implement the preferred `audioPlayerEndInterruption:withFlags:` method, it will be called instead of this one.

When an interruption ends, such as by a user ignoring an incoming phone call, the audio session for your application is automatically reactivated; at that point you can again interact with the audio player. To resume playback, call the `play` (page 79) method.

Availability

Available in iOS 2.2 and later.

See Also

- `audioPlayerBeginInterruption:` (page 288)
- `audioPlayerEndInterruption:withFlags:` (page 289)

Declared In

AVAudioPlayer.h

audioPlayerEndInterruption:withFlags:

Called after your audio session interruption ends, with flags indicating the state of the audio session.

```
- (void)audioPlayerEndInterruption:(AVAudioPlayer *)player
    withFlags:(NSUInteger)flags
```

Parameters

player

The audio player whose interruption has ended.

flags

Flags indicating the state of the audio session when this method is called. Flags are described in `Interruption Flags` (page 100).

Discussion

When an interruption ends, such as by a user ignoring an incoming phone call, the audio session for your application is automatically reactivated; at that point you can again interact with the audio player. To resume playback, call the `play` (page 79) method.

If this delegate method receives the `AVAudioSessionInterruptionFlags_ShouldResume` (page 100) constant in its *flags* parameter, the audio session is immediately ready to be used.

If you implement this method, the system does not call the `audioPlayerEndInterruption:` (page 289) method.

Availability

Available in iOS 4.0 and later.

See Also

`- audioPlayerBeginInterruption:` (page 288)

Declared In

`AVAudioPlayer.h`

AVAudioRecorderDelegate Protocol Reference

Conforms to	NSObject
Framework	/System/Library/Frameworks/AVFoundation.framework
Availability	Available in iOS 3.0 and later.
Declared in	

Overview

The delegate of an `AVAudioRecorder` object must adopt the `AVAudioRecorderDelegate` protocol. All of the methods in this protocol are optional. They allow a delegate to respond to audio interruptions and audio decoding errors, and to the completion of a recording.

Tasks

Responding to the Completion of a Recording

`- audioRecorderDidFinishRecording:successfully:` (page 292)
Called by the system when a recording is stopped or has finished due to reaching its time limit.

Responding to an Audio Encoding Error

`- audioRecorderEncodeErrorDidOccur:error:` (page 292)
Called when an audio recorder encounters an encoding error during recording.

Handling Audio Interruptions

`- audioRecorderBeginInterruption:` (page 292)
Called when the audio session is interrupted during a recording, such as by an incoming phone call.

`- audioRecorderEndInterruption:` (page 293)
Called after your audio session interruption ends.

`- audioRecorderEndInterruption:withFlags:` (page 293)
Called after your audio session interruption ends, with flags indicating the state of the audio session.

Instance Methods

audioRecorderBeginInterruption:

Called when the audio session is interrupted during a recording, such as by an incoming phone call.

```
- (void)audioRecorderBeginInterruption:(AVAudioRecorder *)recorder
```

Parameters

recorder

The audio recorder whose recording was interrupted.

Discussion

Upon interruption, your application's audio session is deactivated and the audio recorder pauses. You cannot use the audio recorder again until you receive a notification that the interruption has ended.

Availability

Available in iOS 3.0 and later.

See Also

- audioRecorderEndInterruption:withFlags: (page 293)

Declared In

AVAudioRecorder.h

audioRecorderDidFinishRecording:successfully:

Called by the system when a recording is stopped or has finished due to reaching its time limit.

```
- (void)audioRecorderDidFinishRecording:(AVAudioRecorder *)recorder
    successfully:(BOOL)flag
```

Parameters

recorder

The audio recorder that has finished recording.

flag

TRUE on successful completion of recording; FALSE if recording stopped because of an audio encoding error.

Discussion

This method is not called by the system if the audio recorder stopped due to an interruption.

Availability

Available in iOS 3.0 and later.

Declared In

AVAudioRecorder.h

audioRecorderEncodeErrorDidOccur:error:

Called when an audio recorder encounters an encoding error during recording.

```
- (void)audioRecorderEncodeErrorDidOccur:(AVAudioRecorder *)recorder
    error:(NSError *)error
```

Parameters

recorder

The audio recorder that encountered the encoding error.

error

The encoding error.

Availability
Available in iOS 3.0 and later.

Declared In
AVAudioRecorder.h

audioRecorderEndInterruption:

Called after your audio session interruption ends.

```
- (void)audioRecorderEndInterruption:(AVAudioRecorder *)recorder
```

Parameters

recorder

The paused audio recorder whose interruption has ended.

Discussion
If you implement the preferred `audioRecorderEndInterruption:withFlags:` method, it will be called instead of this one.

For an audio recorder's delegate to receive this message, the audio recorder must have been recording when the interruption started. When an interruption ends, such as by a user ignoring an incoming phone call, the audio session for your application is automatically reactivated; at that point you can again interact with the audio recorder. To resume recording, call the `record` (page 89) method.

Availability
Available in iOS 3.0 and later.

See Also
- `audioRecorderBeginInterruption:` (page 292)
- `audioRecorderEndInterruption:withFlags:` (page 293)

Declared In
AVAudioRecorder.h

audioRecorderEndInterruption:withFlags:

Called after your audio session interruption ends, with flags indicating the state of the audio session.

```
- (void)audioRecorderEndInterruption:(AVAudioRecorder *)recorder
    withFlags:(NSUInteger)flags
```

Parameters

recorder

The paused audio recorder whose interruption has ended.

flags

Flags indicating the state of the audio session when this method is called. Flags are described in `Interruption Flags` (page 100).

Discussion

For an audio recorder's delegate to receive this message, the audio recorder must have been recording when the interruption started. When an interruption ends, such as by a user ignoring an incoming phone call, the audio session for your application is automatically reactivated; at that point you can again interact with the audio recorder. To resume recording, call the `record` (page 89) method.

If this delegate method receives the `AVAudioSessionInterruptionFlags_ShouldResume` (page 100) constant in its *flags* parameter, the audio session is immediately ready to be used.

If you implement this method, the system does not call the `audioRecorderEndInterruption:` (page 293) method.

Availability

Available in iOS 4.0 and later.

See Also

`- audioRecorderBeginInterruption:` (page 292)

Declared In

`AVAudioRecorder.h`

AVAudioSessionDelegate Protocol Reference

Conforms to	NSObject
Framework	/System/Library/Frameworks/AVFoundation.framework/
Availability	Available in iOS 3.0 and later.
Declared in	
Companion guide	Audio Session Programming Guide

Overview

The delegate of an `AVAudioSession` object must adopt the `AVAudioSessionDelegate` protocol. The methods in this protocol are optional. They allow a delegate to respond to the following sorts of changes in state:

- Changes to the availability of audio input
- Audio session interruption, or end of audio session interruption

An `AVAudioSession` delegate can respond to interruptions at the audio session level. You can use this interface along with any iOS audio technology. For example, your `AVAudioSession` delegate can handle interruptions for OpenAL and audio unit playback.

When using the AV Foundation framework for recording or playback, you can also respond to interruptions at the individual recorder or player level. To do this, create audio recorder or audio player delegates using the protocols described in *AVAudioRecorderDelegate Protocol Reference* and *AVAudioPlayerDelegate Protocol Reference*.

Tasks

Delegate Methods

`- beginInterruption` (page 296)
Called after your audio session is interrupted.

`- endInterruption` (page 296)
Called after your audio session interruption ends.

`- endInterruptionWithFlags:` (page 297)
Called after your audio session interruption ends, with flags indicating the state of the audio session.

- `inputIsAvailableChanged:` (page 297)
  Called after the availability of audio input changes on a device.

Instance Methods

beginInterruption

Called after your audio session is interrupted.

```
- (void)beginInterruption
```

Discussion
By the time this interruption arrives, your audio has already stopped. Your application may be suspended or terminated following an interruption—for example, if a user chooses to take an incoming phone call. Use this method to adjust the user interface, and to save application state, as necessary.

Availability
Available in iOS 3.0 and later.

See Also
- `endInterruption` (page 296)
- `endInterruptionWithFlags:` (page 297)

Declared In
`AVAudioSession.h`

endInterruption

Called after your audio session interruption ends.

```
- (void)endInterruption
```

Discussion
The `endInterruptionWithFlags:` (page 297) method provides you with more information upon interruption end than this method does. Apple recommends that you use `endInterruptionWithFlags:` instead of this method.

If you implement the `endInterruptionWithFlags:` (page 297) method, that method is called instead of this one when an interruption ends.

To resume using audio after an interruption ends, you must ensure that your audio session is active. `AVAudioPlayer` and `AVAudioRecorder` instances reactivate your audio session automatically when an interruption ends. If you are using another audio technology, such as OpenAL, audio units, or audio queues, you must reactivate your audio session yourself before you can again use audio.

You can also use this method to update the user interface and application state, as necessary.

Availability
Available in iOS 3.0 and later.

See Also
- `beginInterruption` (page 296)
- `endInterruptionWithFlags:` (page 297)

Declared In
`AVAudioSession.h`

endInterruptionWithFlags:

Called after your audio session interruption ends, with flags indicating the state of the audio session.

```
- (void)endInterruptionWithFlags:(NSUInteger)flags
```

Parameters

flags

Flags indicating the state of the audio session when this method is called. Flags are described in `Interruption Flags` (page 100).

Discussion

To resume using audio after an interruption ends, you must ensure that your audio session is active. `AVAudioPlayer` and `AVAudioRecorder` instances reactivate your audio session automatically when an interruption ends. If you are using another audio technology, such as OpenAL, audio units, or audio queues, you must reactivate your audio session yourself before you can again use audio.

You can also use this method to update the user interface and application state, as necessary.

If this delegate method receives the `AVAudioSessionInterruptionFlags_ShouldResume` (page 100) constant in its *flags* parameter, the audio session is immediately ready to be used.

If you implement this method, it is called instead of the `endInterruption` (page 296) method when an interruption ends.

Availability

Available in iOS 4.0 and later.

See Also
- `beginInterruption` (page 296)
- `endInterruption` (page 296)

Declared In
`AVAudioSession.h`

inputIsAvailableChanged:

Called after the availability of audio input changes on a device.

```
- (void)inputIsAvailableChanged:(BOOL)isInputAvailable
```

Parameters

isInputAvailable

`YES` if audio input is now available, or `NO` if it is not.

Availability
Available in iOS 3.0 and later.

Declared In
`AVAudioSession.h`

AVCaptureAudioDataOutputSampleBufferDelegate Protocol Reference

Conforms to	NSObject
Framework	/System/Library/Frameworks/AVFoundation.framework/
Availability	Available in iOS 4.0 and later.
Declared in	AVCaptureOutput.h

Overview

The delegate of an `AVCaptureAudioDataOutputSampleBuffer` object must adopt the `AVCaptureAudioDataOutputSampleBufferDelegate` protocol. The method in this protocol is optional.

Tasks

Delegate Methods

`- captureOutput:didOutputSampleBuffer:fromConnection:` (page 299) *required method*

Notifies the delegate that a sample buffer was written. (required)

Instance Methods

captureOutput:didOutputSampleBuffer:fromConnection:

Notifies the delegate that a sample buffer was written. (required)

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

Parameters

captureOutput

The capture output object.

sampleBuffer

The sample buffer that was output.

connection

The connection.

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureOutput.h`

AVCaptureFileOutputRecordingDelegate Protocol Reference

Adopted by	Delegate of an `AVCaptureAudioDataOutput` object.
Conforms to	NSObject
Framework	/System/Library/Frameworks/AVFoundation.framework/
Availability	Available in iOS 4.0 and later.
Declared in	AVFoundation/AVCaptureOutput.h

Overview

The delegate of an `AVCaptureFileOutput` object must adopt the `AVCaptureFileOutputRecordingDelegate` protocol. The methods in this protocol are optional.

Tasks

Delegate Methods

- `captureOutput:didStartRecordingToOutputFileAtURL:fromConnections:` (page 302)
  Called when the capture object starts saving data to a file.
- `captureOutput:didFinishRecordingToOutputFileAtURL:fromConnections:error:` (page 301)
  Called when the capture object stops writing data.

Instance Methods

captureOutput:didFinishRecordingToOutputFileAtURL:fromConnections:error:

Called when the capture object stops writing data.

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

Parameters

captureOutput
The capture output object.

outputFileURL
The output file location.

connections
The connections producing the output.

error
If the file was not written successfully, an error object that describes the problem; otherwise `nil`.

Discussion
This method is called whenever a file is finished. If the file was forced to be finished due to an error, the error is described in the error parameter. Otherwise, the error parameter is `nil`.

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureOutput.h`

captureOutput:didStartRecordingToOutputFileAtURL:fromConnections:

Called when the capture object starts saving data to a file.

```
- (void)captureOutput:(AVCaptureFileOutput *)captureOutput
    didStartRecordingToOutputFileAtURL:(NSURL *)fileURL
    fromConnections:(NSArray *)connections
```

Parameters

captureOutput
The capture output object.

fileURL
The output file location.

connections
The connections producing the output.

Availability
Available in iOS 4.0 and later.

Declared In
`AVCaptureOutput.h`

Functions

AV Foundation Functions Reference

Framework: AVFoundation/AVFoundation.h

Overview

This chapter describes the function defined in the AVFoundation Framework.

Functions

AVMakeRectWithAspectRatioInsideRect

Returns a scaled `CGRect` that maintains the aspect ratio specified by a `CGSize` within a bounding `CGRect`.

```
CGRect AVMakeRectWithAspectRatioInsideRect(CGSize aspectRatio, CGRect boundingRect);
```

Parameters

aspectRatio

The width and height ratio (aspect ratio) you want to maintain.

boundingRect

The bounding rectangle you want to fit into.

Return Value

Returns a scaled `CGRect` that maintains the aspect ratio specified by *aspectRatio* that fits within *boundingRect*.

Discussion

This is useful when attempting to fit the `naturalSize` property of an `AVPlayerItem` object within the bounds of another `CALayer`. You would typically use the return value of this function as an `AVPlayerLayer` frame property value. For example:

```
myPlayerLayer.frame =
AVMakeRectWithAspectRatioInsideRect(myPlayerItem.naturalSize,
mySuperLayer.bounds);
```

Availability

Available in iOS 4.0 and later.

Declared In

`AVUtilities.h`

Constants

PART IV

Constants

AV Foundation Audio Settings Constants

Framework: AVFoundation/AVAudioSettings.h

Declared in

Overview

Use these audio settings keys to configure an `AVAudioRecorder` object. You can also use some of these keys to retrieve information about the sound associated with an `AVAudioPlayer` object, such as audio data format, sample rate, and number of channels.

Note: The constants described in this document were previously described in *AVAudioRecorder Class Reference*.

Constants

General Audio Format Settings

Audio settings that apply to all audio formats handled by the `AVAudioPlayer` and `AVAudioRecorder` classes.

```
NSString *const AVFormatIDKey;
NSString *const AVSampleRateKey;
NSString *const AVNumberOfChannelsKey;
```

Constants

`AVFormatIDKey`

A format identifier. See the "Audio Data Format Identifiers" enumeration in *Core Audio Data Types Reference*.

Available in iOS 3.0 and later.

Declared in `AVAudioSettings.h`.

`AVSampleRateKey`

A sample rate, in hertz, expressed as an `NSNumber` floating point value.

Available in iOS 3.0 and later.

Declared in `AVAudioSettings.h`.

`AVNumberOfChannelsKey`

The number of channels expressed as an `NSNumber` integer value.

Available in iOS 3.0 and later.

Declared in `AVAudioSettings.h`.

Linear PCM Format Settings

Audio settings that apply to linear PCM audio formats.

```
NSString *const AVLinearPCMBitDepthKey;
NSString *const AVLinearPCMIsBigEndianKey;
NSString *const AVLinearPCMIsFloatKey;
NSString *const AVLinearPCMIsNonInterleaved;
```

Constants

`AVLinearPCMBitDepthKey`

An `NSNumber` integer that indicates the bit depth for a linear PCM audio format—one of 8, 16, 24, or 32.

Available in iOS 3.0 and later.

Declared in `AVAudioSettings.h`.

`AVLinearPCMIsBigEndianKey`

A Boolean value that indicates whether the audio format is big endian (`YES`) or little endian (`NO`).

Available in iOS 3.0 and later.

Declared in `AVAudioSettings.h`.

`AVLinearPCMIsFloatKey`

A Boolean value that indicates that the audio format is floating point (`YES`) or fixed point (`NO`).

Available in iOS 3.0 and later.

Declared in `AVAudioSettings.h`.

`AVLinearPCMIsNonInterleaved`

A Boolean value that indicates that the audio format is non-interleaved (`YES`) or interleaved (`NO`).

Available in iOS 4.0 and later.

Declared in `AVAudioSettings.h`.

Encoder Settings

Audio encoder settings for the `AVAudioRecorder` class.

```
NSString *const AVEncoderAudioQualityKey;
NSString *const AVEncoderBitRateKey;
NSString *const AVEncoderBitRatePerChannelKey;
NSString *const AVEncoderBitDepthHintKey;
```

Constants

`AVEncoderAudioQualityKey`

A constant from "Audio Quality Flags" (page 311).

Available in iOS 3.0 and later.

Declared in `AVAudioSettings.h`.

`AVEncoderBitRateKey`

An integer that identifies the audio bit rate.

Available in iOS 3.0 and later.

Declared in `AVAudioSettings.h`.

`AVEncoderBitRatePerChannelKey`
An integer that identifies the audio bit rate per channel.

Available in iOS 4.0 and later.

Declared in `AVAudioSettings.h`.

`AVEncoderBitDepthHintKey`
An integer ranging from 8 through 32.

Available in iOS 3.0 and later.

Declared in `AVAudioSettings.h`.

Sample Rate Conversion Settings

Sample rate converter audio quality settings.

```
NSString *const AVSampleRateConverterAudioQualityKey;
```

Constants

`AVSampleRateConverterAudioQualityKey`
An `NSNumber` integer value. See "Audio Quality Flags" (page 311).

Available in iOS 3.0 and later.

Declared in `AVAudioSettings.h`.

Channel Layout Keys

Key to retrieve channel layout information for playback.

```
NSString *const AVChannelLayoutKey;
```

Constants

`AVChannelLayoutKey`
The corresponding value is an `NSData` object containing an `AudioChannelLayout` structure.

Available in iOS 4.0 and later.

Declared in `AVAudioSettings.h`.

Sample Rate Conversion Audio Quality Flags

Keys that specify sample rate conversion quality, used for the `AVSampleRateConverterAudioQualityKey` (page 311) property.

```
enum {
    AVAudioQualityMin       = 0,
    AVAudioQualityLow       = 0x20,
    AVAudioQualityMedium    = 0x40,
    AVAudioQualityHigh      = 0x60,
    AVAudioQualityMax       = 0x7F
};
typedef NSInteger AVAudioQuality;
```

Constants

`AVAudioQualityMin`

The minimum quality for sample rate conversion.

Available in iOS 3.0 and later.

Declared in `AVAudioSettings.h`.

`AVAudioQualityLow`

Low quality rate conversion.

Available in iOS 3.0 and later.

Declared in `AVAudioSettings.h`.

`AVAudioQualityMedium`

Medium quality sample rate conversion.

Available in iOS 3.0 and later.

Declared in `AVAudioSettings.h`.

`AVAudioQualityHigh`

High quality sample rate conversion.

Available in iOS 3.0 and later.

Declared in `AVAudioSettings.h`.

`AVAudioQualityMax`

Maximum quality sample rate conversion.

Available in iOS 3.0 and later.

Declared in `AVAudioSettings.h`.

AV Foundation Constants Reference

Framework:	AVFoundation/AVFoundation.h
Declared in	AVVideoSettings.h AVAnimation.h AVMediaFormat.h

Overview

This document describes constants defined in the AV Foundation framework not described in individual classes or in domain-specific constants references. See also:

- *AV Foundation Audio Settings Constants*
- *AV Foundation Error Constants*
- *AV Foundation ID3 Constants*
- *AV Foundation iTunes Metadata Constants*
- *AV Foundation QuickTime Constants*

Constants

Media Types

Constants to identify various media types.

```
NSString *const AVMediaTypeVideo;
NSString *const AVMediaTypeAudio;
NSString *const AVMediaTypeText;
NSString *const AVMediaTypeClosedCaption;
NSString *const AVMediaTypeSubtitle;
NSString *const AVMediaTypeTimecode;
NSString *const AVMediaTypeTimedMetadata;
NSString *const AVMediaTypeMuxed;
```

Constants

`AVMediaTypeVideo`

Specifies video.

Available in iOS 4.0 and later.

Declared in `AVMediaFormat.h`.

`AVMediaTypeAudio`
Specifies audio.

Available in iOS 4.0 and later.

Declared in `AVMediaFormat.h`.

`AVMediaTypeText`
Specifies text.

Available in iOS 4.0 and later.

Declared in `AVMediaFormat.h`.

`AVMediaTypeClosedCaption`
Specifies closed-caption content.

Available in iOS 4.0 and later.

Declared in `AVMediaFormat.h`.

`AVMediaTypeSubtitle`
Specifies subtitles.

Available in iOS 4.0 and later.

Declared in `AVMediaFormat.h`.

`AVMediaTypeTimecode`
Specifies a time code.

Available in iOS 4.0 and later.

Declared in `AVMediaFormat.h`.

`AVMediaTypeTimedMetadata`
Specifies timed metadata.

Available in iOS 4.0 and later.

Declared in `AVMediaFormat.h`.

`AVMediaTypeMuxed`
Specifies muxed media.

Available in iOS 4.0 and later.

Declared in `AVMediaFormat.h`.

Video Gravity

These string constants define how the video is displayed within a layer's bounds rectangle.

```
NSString * const AVLayerVideoGravityResize;
NSString * const AVLayerVideoGravityResizeAspect;
NSString * const AVLayerVideoGravityResizeAspectFill;
```

Constants

`AVLayerVideoGravityResize`
Specifies that the video should be stretched to fill the layer's bounds.

Available in iOS 4.0 and later.

Declared in `AVAnimation.h`.

`AVLayerVideoGravityResizeAspect`
Specifies that the player should preserve the video's aspect ratio and fit the video within the layer's bounds.

Available in iOS 4.0 and later.

Declared in `AVAnimation.h`.

`AVLayerVideoGravityResizeAspectFill`
Specifies that the player should preserve the video's aspect ratio and fill the layer's bounds.

Available in iOS 4.0 and later.

Declared in `AVAnimation.h`.

Discussion
You use these constants when setting the `videoGravity` property of an `AVPlayerLayer` or `AVCaptureVideoPreviewLayer` instance.

Media Characteristics

Constants to specify the characteristics of media types.

```
NSString *const AVMediaCharacteristicVisual;
NSString *const AVMediaCharacteristicAudible;
NSString *const AVMediaCharacteristicLegible;
NSString *const AVMediaCharacteristicFrameBased;
```

Constants
`AVMediaCharacteristicVisual`
Indicates that the media is visual.

Available in iOS 4.0 and later.

Declared in `AVMediaFormat.h`.

`AVMediaCharacteristicAudible`
Indicates that the media is audible.

Available in iOS 4.0 and later.

Declared in `AVMediaFormat.h`.

`AVMediaCharacteristicLegible`
Indicates that the media is legible.

Available in iOS 4.0 and later.

Declared in `AVMediaFormat.h`.

`AVMediaCharacteristicFrameBased`
Indicates that the media is frame-based.

Available in iOS 4.0 and later.

Declared in `AVMediaFormat.h`.

Video Settings

These constants define dictionary keys for configuring video compression and compression settings for video assets.

```
NSString *const AVVideoCodecKey;
NSString *const AVVideoCodecH264;
NSString *const AVVideoCodecJPEG;
NSString *const AVVideoWidthKey;
NSString *const AVVideoHeightKey;
NSString *const AVVideoCompressionPropertiesKey;
NSString *const AVVideoAverageBitRateKey;
NSString *const AVVideoMaxKeyFrameIntervalKey;
NSString *const AVVideoProfileLevelKey;
NSString *const AVVideoProfileLevelH264Baseline30;
NSString *const AVVideoProfileLevelH264Baseline31;
NSString *const AVVideoProfileLevelH264Main30;
NSString *const AVVideoProfileLevelH264Main31;
NSString *const AVVideoPixelAspectRatioKey;
NSString *const AVVideoPixelAspectRatioHorizontalSpacingKey;
NSString *const AVVideoPixelAspectRatioVerticalSpacingKey;
NSString *const AVVideoCleanApertureKey;
NSString *const AVVideoCleanApertureWidthKey;
NSString *const AVVideoCleanApertureHeightKey;
NSString *const AVVideoCleanApertureHorizontalOffsetKey;
NSString *const AVVideoCleanApertureVerticalOffsetKey;
```

Constants

`AVVideoCodecKey`

Specifies a key to access the name of the codec used to encode the video.

The corresponding value is an instance of `NSString`; equivalent to `CMVideoCodecType`.

Available in iOS 4.0 and later.

Declared in `AVVideoSettings.h`.

`AVVideoCodecH264`

Specifies that the video was encoded using H264.

Available in iOS 4.0 and later.

Declared in `AVVideoSettings.h`.

`AVVideoCodecJPEG`

Specifies that the video was encoded using the JPEG encoder.

Available in iOS 4.0 and later.

Declared in `AVVideoSettings.h`.

`AVVideoWidthKey`

Specifies a key to access the width of the video in pixels.

The corresponding value is an instance of `NSNumber`.

Available in iOS 4.0 and later.

Declared in `AVVideoSettings.h`.

`AVVideoHeightKey`

Specifies a key to access the height of the video in pixels.

The corresponding value is an instance of `NSNumber`.

Available in iOS 4.0 and later.

Declared in `AVVideoSettings.h`.

`AVVideoCompressionPropertiesKey`

Specifies a key to access the compression properties.

The corresponding value is an instance of `NSDictionary`.

Available in iOS 4.0 and later.

Declared in `AVVideoSettings.h`.

`AVVideoAverageBitRateKey`

Specifies a key to access the average bit rate (as bits per second) used in encoding.

The corresponding value is an instance of `NSNumber`.

Available in iOS 4.0 and later.

Declared in `AVVideoSettings.h`.

`AVVideoMaxKeyFrameIntervalKey`

Specifies a key to access the maximum interval between key frames.

The corresponding value is an instance of `NSNumber`. `1` means key frames only.

Available in iOS 4.0 and later.

Declared in `AVVideoSettings.h`.

`AVVideoProfileLevelKey`

Specifies a key to access the video profile.

Available in iOS 4.0 and later.

Declared in `AVVideoSettings.h`.

`AVVideoProfileLevelH264Baseline30`

Specifies a baseline level 3.0 profile.

Available in iOS 4.0 and later.

Declared in `AVVideoSettings.h`.

`AVVideoProfileLevelH264Baseline31`

Specifies a baseline level 3.1 profile.

Available in iOS 4.0 and later.

Declared in `AVVideoSettings.h`.

`AVVideoProfileLevelH264Main30`

Specifies a main level 3.0 profile.

Available in iOS 4.0 and later.

Declared in `AVVideoSettings.h`.

`AVVideoProfileLevelH264Main31`

Specifies a main level 3.0 profile.

Available in iOS 4.0 and later.

Declared in `AVVideoSettings.h`.

`AVVideoPixelAspectRatioKey`

Specifies a key to access the pixel aspect ratio.

The corresponding value is an instance of `NSDictionary`.

Available in iOS 4.0 and later.

Declared in `AVVideoSettings.h`.

`AVVideoPixelAspectRatioHorizontalSpacingKey`
Specifies a key to access the pixel aspect ratio horizontal spacing.

The corresponding value is an instance of `NSNumber`.

Available in iOS 4.0 and later.

Declared in `AVVideoSettings.h`.

`AVVideoPixelAspectRatioVerticalSpacingKey`
Specifies a key to access the pixel aspect ratio vertical spacing.

The corresponding value is an instance of `NSNumber`.

Available in iOS 4.0 and later.

Declared in `AVVideoSettings.h`.

`AVVideoCleanApertureKey`
Specifies a key to access the clean aperture.

The corresponding value is an instance of `NSDictionary`.

Available in iOS 4.0 and later.

Declared in `AVVideoSettings.h`.

`AVVideoCleanApertureWidthKey`
Specifies a key to access the clean aperture width.

The corresponding value is an instance of `NSNumber`.

Available in iOS 4.0 and later.

Declared in `AVVideoSettings.h`.

`AVVideoCleanApertureHeightKey`
Specifies a key to access the clean aperture height.

The corresponding value is an instance of `NSNumber`.

Available in iOS 4.0 and later.

Declared in `AVVideoSettings.h`.

`AVVideoCleanApertureHorizontalOffsetKey`
Specifies a key to access the clean aperture horizontal offset.

The corresponding value is an instance of `NSNumber`.

Available in iOS 4.0 and later.

Declared in `AVVideoSettings.h`.

`AVVideoCleanApertureVerticalOffsetKey`
Specifies a key to access the clean aperture vertical offset.

The corresponding value is an instance of `NSNumber`.

Available in iOS 4.0 and later.

Declared in `AVVideoSettings.h`.

File Format UTIs

These constants specify UTIs for various file formats.

```
NSString *const AVFileType3GPP;
NSString *const AVFileTypeAIFC;
NSString *const AVFileTypeAIFF;
NSString *const AVFileTypeAMR;
NSString *const AVFileTypeCoreAudioFormat;
NSString *const AVFileTypeAppleM4V;
NSString *const AVFileTypeMPEG4;
NSString *const AVFileTypeAppleM4A;
NSString *const AVFileTypeQuickTimeMovie;
NSString *const AVFileTypeWAVE;
```

Constants

`AVFileType3GPP`

UTI for the 3GPP file format.

The value of this UTI is `public.3gpp`. Files are identified with the .3gp, .3gpp, and .sdv extensions.

Available in iOS 4.0 and later.

Declared in `AVMediaFormat.h`.

`AVFileTypeAIFC`

UTI for the AIFC audio file format.

The value of this UTI is `public.aifc-audio`. Files are identified with the .aifc and .cdda extensions.

Available in iOS 4.0 and later.

Declared in `AVMediaFormat.h`.

`AVFileTypeAIFF`

UTI for the AIFF audio file format.

The value of this UTI is `public.aiff-audio`. Files are identified with the .aif and .aiff extensions.

Available in iOS 4.0 and later.

Declared in `AVMediaFormat.h`.

`AVFileTypeCoreAudioFormat`

UTI for the CoreAudio file format.

The value of this UTI is `com.apple.coreaudio-format`. Files are identified with the .caf extension.

Available in iOS 4.0 and later.

Declared in `AVMediaFormat.h`.

`AVFileTypeAppleM4V`

UTI for the iTunes video file format.

The value of this UTI is `com.apple.mpeg-4-video`. Files are identified with the .m4v extension.

Available in iOS 4.0 and later.

Declared in `AVMediaFormat.h`.

`AVFileTypeMPEG4`

UTI for the MPEG-4 file format.

The value of this UTI is `public.mpeg-4`. Files are identified with the .mp4 extension.

Available in iOS 4.0 and later.

Declared in `AVMediaFormat.h`.

`AVFileTypeAppleM4A`
: UTI for the Apple m4a audio file format.

  The value of this UTI is `com.apple.m4a-audio`. Files are identified with the .m4a extension.

  Available in iOS 4.0 and later.

  Declared in `AVMediaFormat.h`.

`AVFileTypeQuickTimeMovie`
: UTI for the QuickTime movie file format.

  The value of this UTI is `com.apple.quicktime-movie`. Files are identified with the .mov and .qt extensions.

  Available in iOS 4.0 and later.

  Declared in `AVMediaFormat.h`.

`AVFileTypeWAVE`
: UTI for the QuickTime movie file format.

  The value of this UTI is `com.microsoft.waveform-audio`. Files are identified with the .wav, .wave, and .bwf extensions.

  Available in iOS 4.0 and later.

  Declared in `AVMediaFormat.h`.

`AVFileTypeAMR`
: UTI for the adaptive multi-rate audio file format.

  The value of this UTI is `org.3gpp.adaptive-multi-rate-audio`. Files are identified with the .amr extension.

  Available in iOS 4.0 and later.

  Declared in `AVMediaFormat.h`.

Core Animation

Support for integration with Core Animation.

```
const CFTimeInterval AVCoreAnimationBeginTimeAtZero
```

Constants

`AVCoreAnimationBeginTimeAtZero`
: Use this constant to set the CoreAnimation's animation `beginTime` property to be time `0`.

  The constant is a small, non-zero, positive value which prevents CoreAnimation from replacing `0.0` with `CACurrentMediaTime`.

  Available in iOS 4.0 and later.

  Declared in `AVAnimation.h`.

Metadata Keys

Common metadata and common keys for metadata.

```
NSString *const AVMetadataKeySpaceCommon;
```

```
NSString *const AVMetadataCommonKeyTitle;
NSString *const AVMetadataCommonKeyCreator;
NSString *const AVMetadataCommonKeySubject;
NSString *const AVMetadataCommonKeyDescription;
NSString *const AVMetadataCommonKeyPublisher;
NSString *const AVMetadataCommonKeyContributor;
NSString *const AVMetadataCommonKeyCreationDate;
NSString *const AVMetadataCommonKeyLastModifiedDate;
NSString *const AVMetadataCommonKeyType;
NSString *const AVMetadataCommonKeyFormat;
NSString *const AVMetadataCommonKeyIdentifier;
NSString *const AVMetadataCommonKeySource;
NSString *const AVMetadataCommonKeyLanguage;
NSString *const AVMetadataCommonKeyRelation;
NSString *const AVMetadataCommonKeyLocation;
NSString *const AVMetadataCommonKeyCopyrights;
NSString *const AVMetadataCommonKeyAlbumName;
NSString *const AVMetadataCommonKeyAuthor;
NSString *const AVMetadataCommonKeyArtist;
NSString *const AVMetadataCommonKeyArtwork;
NSString *const AVMetadataCommonKeyMake;
NSString *const AVMetadataCommonKeyModel;
NSString *const AVMetadataCommonKeySoftware;
```

Constants

`AVMetadataKeySpaceCommon`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataCommonKeyTitle`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataCommonKeyCreator`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataCommonKeySubject`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataCommonKeyDescription`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataCommonKeyPublisher`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataCommonKeyContributor`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataCommonKeyCreationDate`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataCommonKeyLastModifiedDate`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataCommonKeyType`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataCommonKeyFormat`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataCommonKeyIdentifier`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataCommonKeySource`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataCommonKeyLanguage`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataCommonKeyRelation`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataCommonKeyLocation`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataCommonKeyCopyrights`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataCommonKeyAlbumName`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataCommonKeyAuthor`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataCommonKeyArtist`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataCommonKeyArtwork`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataCommonKeyMake`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataCommonKeyModel`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataCommonKeySoftware`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

AV Foundation Error Constants

Framework: AVFoundation/AVFoundation.h

Declared in AVError.h

Overview

This document describes the error constants defined in the AV Foundation framework not described in individual classes.

Constants

Error Domain

Constant to identify the AVFoundation error domain.

```
const NSString *AVFoundationErrorDomain;
```

Constants

`AVFoundationErrorDomain`

Domain for AVFoundation errors.

Available in iOS 4.0 and later.

Declared in `AVError.h`.

Error User Info Keys

Keys in the user info dictionary in errors AVFoundation creates.

```
NSString *const AVErrorDeviceKey;
NSString *const AVErrorExcludingDeviceKey;
NSString *const AVErrorTimeKey;
NSString *const AVErrorFileSizeKey;
NSString *const AVErrorPIDKey;
NSString *const AVErrorRecordingSuccessfullyFinishedKey;
```

Constants

`AVErrorDeviceKey`

Available in iOS 4.0 and later.

Declared in `AVError.h`.

`AVErrorExcludingDeviceKey`

`AVErrorTimeKey`

Available in iOS 4.0 and later.

Declared in `AVError.h`.

`AVErrorFileSizeKey`

Available in iOS 4.0 and later.

Declared in `AVError.h`.

`AVErrorPIDKey`

Available in iOS 4.0 and later.

Declared in `AVError.h`.

`AVErrorRecordingSuccessfullyFinishedKey`

Available in iOS 4.0 and later.

Declared in `AVError.h`.

General Error Codes

Error codes that denote a general error.

```
enum {
    AVErrorUnknown                                          = -11800,
    AVErrorOutOfMemory                                      = -11801,
    AVErrorSessionNotRunning                                = -11803,
    AVErrorDeviceAlreadyUsedByAnotherSession                = -11804,
    AVErrorNoDataCaptured                                   = -11805,
    AVErrorSessionConfigurationChanged                      = -11806,
    AVErrorDiskFull                                         = -11807,
    AVErrorDeviceWasDisconnected                            = -11808,
    AVErrorMediaChanged                                     = -11809,
    AVErrorMaximumDurationReached                           = -11810,
    AVErrorMaximumFileSizeReached                           = -11811,
    AVErrorMediaDiscontinuity                               = -11812,
    AVErrorMaximumNumberOfSamplesForFileFormatReached       = -11813,
    AVErrorDeviceNotConnected                               = -11814,
    AVErrorDeviceInUseByAnotherApplication                  = -11815,
    AVErrorDeviceLockedForConfigurationByAnotherProcess     = -11817,
    AVErrorSessionWasInterrupted                            = -11818,
    AVErrorMediaServicesWereReset                           = -11819,
    AVErrorExportFailed                                     = -11820,
    AVErrorDecodeFailed                                     = -11821,
    AVErrorInvalidSourceMedia                               = -11822,
    AVErrorFileAlreadyExists                                = -11823,
    AVErrorCompositionTrackSegmentsNotContiguous            = -11824,
    AVErrorInvalidCompositionTrackSegmentDuration           = -11825,
    AVErrorInvalidCompositionTrackSegmentSourceStartTime    = -11826,
    AVErrorInvalidCompositionTrackSegmentSourceDuration     = -11827,
    AVErrorFileFormatNotRecognized                          = -11828,
    AVErrorFileFailedToParse                                = -11829,
    AVErrorMaximumStillImageCaptureRequestsExceeded         = -11830,
    AVErrorContentIsProtected                               = -11831,
    AVErrorNoImageAtTime                                    = -11832,
};
```

Constants

`AVErrorUnknown`

Reason for the error is unknown.

Available in iOS 4.0 and later.

Declared in `AVError.h`.

`AVErrorOutOfMemory`

The operation could not be completed because there is not enough memory to process all of the media.

Available in iOS 4.0 and later.

Declared in `AVError.h`.

`AVErrorSessionNotRunning`

Recording could not be started because no data is being captured.

Available in iOS 4.0 and later.

Declared in `AVError.h`.

`AVErrorDeviceAlreadyUsedByAnotherSession`

Media could not be captured from the device because it is already in use elsewhere in this application.

Available in iOS 4.0 and later.

Declared in `AVError.h`.

`AVErrorNoDataCaptured`
: Recording failed because no data was received.

Available in iOS 4.0 and later.

Declared in `AVError.h`.

`AVErrorSessionConfigurationChanged`
: Recording stopped because the configuration of media sources and destinations changed.

Available in iOS 4.0 and later.

Declared in `AVError.h`.

`AVErrorDiskFull`
: Recording stopped because the disk is getting full.

Available in iOS 4.0 and later.

Declared in `AVError.h`.

`AVErrorDeviceWasDisconnected`
: Recording stopped because the device was turned off or disconnected.

Available in iOS 4.0 and later.

Declared in `AVError.h`.

`AVErrorMediaChanged`
: Recording stopped because the format of the source media changed.

Available in iOS 4.0 and later.

Declared in `AVError.h`.

`AVErrorMaximumDurationReached`
: Recording stopped because the maximum duration for the file was reached.

Available in iOS 4.0 and later.

Declared in `AVError.h`.

`AVErrorMaximumFileSizeReached`
: Recording stopped because the maximum size for the file was reached.

Available in iOS 4.0 and later.

Declared in `AVError.h`.

`AVErrorMediaDiscontinuity`
: Recording stopped because there was an interruption in the input media.

Available in iOS 4.0 and later.

Declared in `AVError.h`.

`AVErrorMaximumNumberOfSamplesForFileFormatReached`
: Recording stopped because the maximum number of samples for the file was reached.

Available in iOS 4.0 and later.

Declared in `AVError.h`.

`AVErrorDeviceNotConnected`
: The device could not be opened because it is not connected or turned on.

Available in iOS 4.0 and later.

Declared in `AVError.h`.

`AVErrorDeviceInUseByAnotherApplication`
: The device could not be opened because it is in use by another application.

  Available in iOS 4.0 and later.

  Declared in `AVError.h`.

`AVErrorDeviceLockedForConfigurationByAnotherProcess`
: Settings for the device could not be changed because the device is being controlled by another application.

  Available in iOS 4.0 and later.

  Declared in `AVError.h`.

`AVErrorSessionWasInterrupted`
: Recording stopped because it was interrupted.

  Available in iOS 4.0 and later.

  Declared in `AVError.h`.

`AVErrorMediaServicesWereReset`
: The operation could not be completed because media services became unavailable.

  Available in iOS 4.0 and later.

  Declared in `AVError.h`.

`AVErrorExportFailed`
: The export could not be completed.

  Available in iOS 4.0 and later.

  Declared in `AVError.h`.

`AVErrorDecodeFailed`
: The operation could not be completed because some source media could not be decoded.

  Available in iOS 4.0 and later.

  Declared in `AVError.h`.

`AVErrorInvalidSourceMedia`
: The operation could not be completed because some source media could not be read.

  Available in iOS 4.0 and later.

  Declared in `AVError.h`.

`AVErrorFileAlreadyExists`
: The file could not be created because a file with the same name already exists in the same location.

  Available in iOS 4.0 and later.

  Declared in `AVError.h`.

`AVErrorCompositionTrackSegmentsNotContiguous`
: The source media can't be added because it contains gaps.

  Available in iOS 4.0 and later.

  Declared in `AVError.h`.

`AVErrorInvalidCompositionTrackSegmentDuration`
: The source media can't be added because its duration in the destination is invalid.

  Available in iOS 4.0 and later.

  Declared in `AVError.h`.

`AVErrorInvalidCompositionTrackSegmentSourceStartTime`
The source media can't be added because its start time in the destination is invalid.

Available in iOS 4.0 and later.

Declared in `AVError.h`.

`AVErrorInvalidCompositionTrackSegmentSourceDuration`
The source media can't be added because it has no duration.

Available in iOS 4.0 and later.

Declared in `AVError.h`.

`AVErrorFileFormatNotRecognized`
The media could not be opened because it is not in a recognized format.

Available in iOS 4.0 and later.

Declared in `AVError.h`.

`AVErrorFileFailedToParse`
The media could not be opened because the file is damaged or not in a recognized format.

Available in iOS 4.0 and later.

Declared in `AVError.h`.

`AVErrorMaximumStillImageCaptureRequestsExceeded`
The photo could not be taken because there are too many photo requests that haven't completed yet.

Available in iOS 4.0 and later.

Declared in `AVError.h`.

`AVErrorContentIsProtected`
The application is not authorized to open the media.

Available in iOS 4.0 and later.

Declared in `AVError.h`.

`AVErrorNoImageAtTime`
There is no image at that time in the media.

Available in iOS 4.0 and later.

Declared in `AVError.h`.

AV Foundation ID3 Constants

Framework: AVFoundation/AVFoundation.h

Declared in AVMetadataFormat.h

Overview

This document describes constants defined in the AV Foundation framework related to ID3 metadata.

Constants

ID3 Metadata Identifiers

ID3 metadata identifiers.

```
NSString *const AVMetadataFormatID3Metadata;
NSString *const AVMetadataKeySpaceID3;
```

Constants

`AVMetadataFormatID3Metadata`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataKeySpaceID3`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

ID3 Metadata Keys

ID3 metadata keys.

```
NSString *const AVMetadataID3MetadataKeyAudioEncryption;
NSString *const AVMetadataID3MetadataKeyAttachedPicture;
NSString *const AVMetadataID3MetadataKeyAudioSeekPointIndex;
NSString *const AVMetadataID3MetadataKeyComments;
NSString *const AVMetadataID3MetadataKeyCommerical;
NSString *const AVMetadataID3MetadataKeyEncryption;
NSString *const AVMetadataID3MetadataKeyEqualization;
NSString *const AVMetadataID3MetadataKeyEqualization2;
NSString *const AVMetadataID3MetadataKeyEventTimingCodes;
NSString *const AVMetadataID3MetadataKeyGeneralEncapsulatedObject;
NSString *const AVMetadataID3MetadataKeyGroupIdentifier;
NSString *const AVMetadataID3MetadataKeyInvolvedPeopleList_v23;
NSString *const AVMetadataID3MetadataKeyLink;
NSString *const AVMetadataID3MetadataKeyMusicCDIdentifier;
NSString *const AVMetadataID3MetadataKeyMPEGLocationLookupTable;
NSString *const AVMetadataID3MetadataKeyOwnership;
NSString *const AVMetadataID3MetadataKeyPrivate;
NSString *const AVMetadataID3MetadataKeyPlayCounter;
NSString *const AVMetadataID3MetadataKeyPopularimeter;
NSString *const AVMetadataID3MetadataKeyPositionSynchronization;
NSString *const AVMetadataID3MetadataKeyRecommendedBufferSize                 /*
RBUF Recommended buffer size */
NSString *const AVMetadataID3MetadataKeyRelativeVolumeAdjustment              /*
RVAD Relative volume adjustment */
NSString *const AVMetadataID3MetadataKeyRelativeVolumeAdjustment2             /*
RVA2 Relative volume adjustment (2) */
NSString *const AVMetadataID3MetadataKeyReverb                                /*
RVRB Reverb */
NSString *const AVMetadataID3MetadataKeySeek                                  /*
SEEK Seek frame */
NSString *const AVMetadataID3MetadataKeySignature                             /*
SIGN Signature frame */
NSString *const AVMetadataID3MetadataKeySynchronizedLyric                     /*
SYLT Synchronized lyric/text */
NSString *const AVMetadataID3MetadataKeySynchronizedTempoCodes                /*
SYTC Synchronized tempo codes */
NSString *const AVMetadataID3MetadataKeyAlbumTitle                            /*
TALB Album/Movie/Show title */
NSString *const AVMetadataID3MetadataKeyBeatsPerMinute                        /*
TBPM BPM (beats per minute) */
NSString *const AVMetadataID3MetadataKeyComposer                              /*
TCOM Composer */
NSString *const AVMetadataID3MetadataKeyContentType                           /*
TCON Content type */
NSString *const AVMetadataID3MetadataKeyCopyright                             /*
TCOP Copyright message */
NSString *const AVMetadataID3MetadataKeyDate                                  /*
TDAT Date */
NSString *const AVMetadataID3MetadataKeyEncodingTime                          /*
TDEN Encoding time */
NSString *const AVMetadataID3MetadataKeyPlaylistDelay                         /*
TDLY Playlist delay */
NSString *const AVMetadataID3MetadataKeyOriginalReleaseTime                   /*
TDOR Original release time */
NSString *const AVMetadataID3MetadataKeyRecordingTime                         /*
TDRC Recording time */
NSString *const AVMetadataID3MetadataKeyReleaseTime                           /*
TDRL Release time */
```

```
NSString *const AVMetadataID3MetadataKeyTaggingTime                    /*
TDTG Tagging time */
NSString *const AVMetadataID3MetadataKeyEncodedBy                      /*
TENC Encoded by */
NSString *const AVMetadataID3MetadataKeyLyricist                       /*
TEXT Lyricist/Text writer */
NSString *const AVMetadataID3MetadataKeyFileType                       /*
TFLT File type */
NSString *const AVMetadataID3MetadataKeyTime                           /*
TIME Time */
NSString *const AVMetadataID3MetadataKeyInvolvedPeopleList_v24         /*
TIPL Involved people list */
NSString *const AVMetadataID3MetadataKeyContentGroupDescription        /*
TIT1 Content group description */
NSString *const AVMetadataID3MetadataKeyTitleDescription               /*
TIT2 Title/songname/content description */
NSString *const AVMetadataID3MetadataKeySubTitle                       /*
TIT3 Subtitle/Description refinement */
NSString *const AVMetadataID3MetadataKeyInitialKey                     /*
TKEY Initial key */
NSString *const AVMetadataID3MetadataKeyLanguage                       /*
TLAN Language(s) */
NSString *const AVMetadataID3MetadataKeyLength                         /*
TLEN Length */
NSString *const AVMetadataID3MetadataKeyMusicianCreditsList            /*
TMCL Musician credits list */
NSString *const AVMetadataID3MetadataKeyMediaType                      /*
TMED Media type */
NSString *const AVMetadataID3MetadataKeyMood                           /*
TMOO Mood */
NSString *const AVMetadataID3MetadataKeyOriginalAlbumTitle             /*
TOAL Original album/movie/show title */
NSString *const AVMetadataID3MetadataKeyOriginalFilename               /*
TOFN Original filename */
NSString *const AVMetadataID3MetadataKeyOriginalLyricist               /*
TOLY Original lyricist(s)/text writer(s) */
NSString *const AVMetadataID3MetadataKeyOriginalArtist                 /*
TOPE Original artist(s)/performer(s) */
NSString *const AVMetadataID3MetadataKeyOriginalReleaseYear            /*
TORY Original release year */
NSString *const AVMetadataID3MetadataKeyFileOwner                      /*
TOWN File owner/licensee */
NSString *const AVMetadataID3MetadataKeyLeadPerformer                  /*
TPE1 Lead performer(s)/Soloist(s) */
NSString *const AVMetadataID3MetadataKeyBand                           /*
TPE2 Band/orchestra/accompaniment */
NSString *const AVMetadataID3MetadataKeyConductor                      /*
TPE3 Conductor/performer refinement */
NSString *const AVMetadataID3MetadataKeyModifiedBy                     /*
TPE4 Interpreted remixed or otherwise modified by */
NSString *const AVMetadataID3MetadataKeyPartOfASet                     /*
TPOS Part of a set */
NSString *const AVMetadataID3MetadataKeyProducedNotice                 /*
TPRO Produced notice */
NSString *const AVMetadataID3MetadataKeyPublisher                      /*
TPUB Publisher */
NSString *const AVMetadataID3MetadataKeyTrackNumber                    /*
TRCK Track number/Position in set */
```

```
NSString *const AVMetadataID3MetadataKeyRecordingDates                          /*
TRDA Recording dates */
NSString *const AVMetadataID3MetadataKeyInternetRadioStationName                /*
TRSN Internet radio station name */
NSString *const AVMetadataID3MetadataKeyInternetRadioStationOwner               /*
TRSO Internet radio station owner */
NSString *const AVMetadataID3MetadataKeySize                                    /*
TSIZ Size */
NSString *const AVMetadataID3MetadataKeyAlbumSortOrder                          /*
TSOA Album sort order */
NSString *const AVMetadataID3MetadataKeyPerformerSortOrder                      /*
TSOP Performer sort order */
NSString *const AVMetadataID3MetadataKeyTitleSortOrder                          /*
TSOT Title sort order */
NSString *const AVMetadataID3MetadataKeyInternationalStandardRecordingCode      /*
TSRC ISRC (international standard recording code) */
NSString *const AVMetadataID3MetadataKeyEncodedWith                             /*
TSSE Software/Hardware and settings used for encoding */
NSString *const AVMetadataID3MetadataKeySetSubtitle                             /*
TSST Set subtitle */
NSString *const AVMetadataID3MetadataKeyYear                                    /*
TYER Year */
NSString *const AVMetadataID3MetadataKeyUserText                                /*
TXXX User defined text information frame */
NSString *const AVMetadataID3MetadataKeyUniqueFileIdentifier                    /*
UFID Unique file identifier */
NSString *const AVMetadataID3MetadataKeyTermsOfUse                              /*
USER Terms of use */
NSString *const AVMetadataID3MetadataKeyUnsynchronizedLyric                     /*
USLT Unsynchronized lyric/text transcription */
NSString *const AVMetadataID3MetadataKeyCommercialInformation                   /*
WCOM Commercial information */
NSString *const AVMetadataID3MetadataKeyCopyrightInformation                    /*
WCOP Copyright/Legal information */
NSString *const AVMetadataID3MetadataKeyOfficialAudioFileWebpage                /*
WOAF Official audio file webpage */
NSString *const AVMetadataID3MetadataKeyOfficialArtistWebpage                   /*
WOAR Official artist/performer webpage */
NSString *const AVMetadataID3MetadataKeyOfficialAudioSourceWebpage              /*
WOAS Official audio source webpage */
NSString *const AVMetadataID3MetadataKeyOfficialInternetRadioStationHomepage    /*
WORS Official Internet radio station homepage */
NSString *const AVMetadataID3MetadataKeyPayment                                 /*
WPAY Payment */
NSString *const AVMetadataID3MetadataKeyOfficialPublisherWebpage                /*
WPUB Publishers official webpage */
NSString *const AVMetadataID3MetadataKeyUserURL                                 /*
WXXX User defined URL link frame */
```

Constants

`AVMetadataID3MetadataKeyAudioEncryption`

AENC audio encryption.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyAttachedPicture`
APIC attached picture.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyAudioSeekPointIndex`
ASPI audio seek point index.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyComments`
COMM comments.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyCommerical`
COMR commercial frame.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyEncryption`
ENCR encryption method registration.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyEqualization`
EQUA equalization.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyEqualization2`
EQU2 equalisation (2).

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyEventTimingCodes`
ETCO event timing codes.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyGeneralEncapsulatedObject`
GEOB general encapsulated object.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyGroupIdentifier`
GRID group identification registration.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyInvolvedPeopleList_v23`
IPLS involved people list.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyLink`
LINK linked information.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyMusicCDIdentifier`
MCDI music CD identifier.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyMPEGLocationLookupTable`
MLLT MPEG location lookup table.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyOwnership`
OWNE ownership frame.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyPrivate`
PRIV private frame.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyPlayCounter`
PCNT play counter.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyPopularimeter`
POPM popularimeter.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyPositionSynchronization`
POSS position synchronisation frame.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyRecommendedBufferSize`
RBUF recommended buffer size.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyRelativeVolumeAdjustment`
RVAD relative volume adjustment.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyRelativeVolumeAdjustment2`
RVA2 relative volume adjustment (2).

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyReverb`
RVRB reverb.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeySeek`
SEEK seek frame.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeySignature`
SIGN signature frame.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeySynchronizedLyric`
SYLT synchronized lyric/text.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeySynchronizedTempoCodes`
SYTC synchronized tempo codes.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyAlbumTitle`
TALB album/Movie/Show title.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyBeatsPerMinute`
TBPM BPM (beats per minute).

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyComposer`
TCOM composer.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyContentType`
TCON content type.
Available in iOS 4.0 and later.
Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyCopyright`
TCOP copyright message.
Available in iOS 4.0 and later.
Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyDate`
TDAT date.
Available in iOS 4.0 and later.
Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyEncodingTime`
TDEN encoding time.
Available in iOS 4.0 and later.
Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyPlaylistDelay`
TDLY playlist delay.
Available in iOS 4.0 and later.
Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyOriginalReleaseTime`
TDOR original release time.
Available in iOS 4.0 and later.
Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyRecordingTime`
TDRC recording time.
Available in iOS 4.0 and later.
Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyReleaseTime`
TDRL release time.
Available in iOS 4.0 and later.
Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyTaggingTime`
TDTG tagging time.
Available in iOS 4.0 and later.
Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyEncodedBy`
TENC encoded by.
Available in iOS 4.0 and later.
Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyLyricist`
TEXT lyricist/text writer.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyFileType`
TFLT file type.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyTime`
TIME time.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyInvolvedPeopleList_v24`
TIPL involved people list.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyContentGroupDescription`
TIT1 content group description.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyTitleDescription`
TIT2 title/songname/content description.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeySubTitle`
TIT3 subtitle/description refinement.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyInitialKey`
TKEY initial key.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyLanguage`
TLAN language(s).

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyLength`
TLEN length.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyMusicianCreditsList`
TMCL musician credits list.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyMediaType`
TMED media type.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyMood`
TMOO mood.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyOriginalAlbumTitle`
TOAL original album/movie/show title.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyOriginalFilename`
TOFN original filename.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyOriginalLyricist`
TOLY original lyricist(s)/text writer(s).

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyOriginalArtist`
TOPE original artist(s)/performer(s).

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyOriginalReleaseYear`
TORY original release year.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyFileOwner`
TOWN file owner/licensee.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyLeadPerformer`
TPE1 lead performer(s)/Soloist(s).

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyBand`
TPE2 band/orchestra/accompaniment.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyConductor`
TPE3 conductor/performer refinement.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyModifiedBy`
TPE4 interpreted, remixed, or otherwise modified by.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyPartOfASet`
TPOS part of a set.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyProducedNotice`
TPRO produced notice.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyPublisher`
TPUB publisher.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyTrackNumber`
TRCK track number/position in set.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyRecordingDates`
TRDA recording dates.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyInternetRadioStationName`
TRSN internet radio station name.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyInternetRadioStationOwner`
TRSO internet radio station owner.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeySize`
TSIZ size.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyAlbumSortOrder`
TSOA album sort order.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyPerformerSortOrder`
TSOP performer sort order.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyTitleSortOrder`
TSOT title sort order.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyInternationalStandardRecordingCode`
TSRC ISRC (international standard recording code).

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyEncodedWith`
TSSE software/hardware and settings used for encoding.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeySetSubtitle`
TSST set subtitle.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyYear`
TYER year.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyUserText`
TXXX user defined text information frame.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyUniqueFileIdentifier`
UFID unique file identifier.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyTermsOfUse`
: USER terms of use.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyUnsynchronizedLyric`
: USLT unsynchronized lyric/text transcription.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyCommercialInformation`
: WCOM commercial information.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyCopyrightInformation`
: WCOP copyright/legal information.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyOfficialAudioFileWebpage`
: WOAF official audio file webpage.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyOfficialArtistWebpage`
: WOAR official artist/performer webpage.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyOfficialAudioSourceWebpage`
: WOAS official audio source webpage.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyOfficialInternetRadioStationHomepage`
: WORS official Internet radio station homepage.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyPayment`
: WPAY payment.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyOfficialPublisherWebpage`
: WPUB publishers official webpage.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataID3MetadataKeyUserURL`

WXXX user defined URL link frame.

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

AV Foundation iTunes Metadata Constants

Framework: AVFoundation/AVFoundation.h

Declared in AVMetadataFormat.h

Overview

This document describes constants defined in the AV Foundation framework that describe iTunes metadata.

Constants

iTunes Metadata

iTunes metadata.

```
NSString *const AVMetadataFormatiTunesMetadata;
NSString *const AVMetadataKeySpaceiTunes;
```

Constants

`AVMetadataFormatiTunesMetadata`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataKeySpaceiTunes`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

iTunes Metadata Keys

iTunes metadata keys.

```
NSString *const AVMetadataiTunesMetadataKeyAlbum;
NSString *const AVMetadataiTunesMetadataKeyArtist;
NSString *const AVMetadataiTunesMetadataKeyUserComment;
NSString *const AVMetadataiTunesMetadataKeyCoverArt;
NSString *const AVMetadataiTunesMetadataKeyCopyright;
NSString *const AVMetadataiTunesMetadataKeyReleaseDate;
NSString *const AVMetadataiTunesMetadataKeyEncodedBy;
NSString *const AVMetadataiTunesMetadataKeyPredefinedGenre;
NSString *const AVMetadataiTunesMetadataKeyUserGenre;
NSString *const AVMetadataiTunesMetadataKeySongName;
NSString *const AVMetadataiTunesMetadataKeyTrackSubTitle;
NSString *const AVMetadataiTunesMetadataKeyEncodingTool;
NSString *const AVMetadataiTunesMetadataKeyComposer;
NSString *const AVMetadataiTunesMetadataKeyAlbumArtist;
NSString *const AVMetadataiTunesMetadataKeyAccountKind;
NSString *const AVMetadataiTunesMetadataKeyAppleID;
NSString *const AVMetadataiTunesMetadataKeyArtistID;
NSString *const AVMetadataiTunesMetadataKeySongID;
NSString *const AVMetadataiTunesMetadataKeyDiscCompilation;
NSString *const AVMetadataiTunesMetadataKeyDiscNumber;
NSString *const AVMetadataiTunesMetadataKeyGenreID;
NSString *const AVMetadataiTunesMetadataKeyGrouping;
NSString *const AVMetadataiTunesMetadataKeyPlaylistID;
NSString *const AVMetadataiTunesMetadataKeyContentRating;
NSString *const AVMetadataiTunesMetadataKeyBeatsPerMin;
NSString *const AVMetadataiTunesMetadataKeyTrackNumber;
NSString *const AVMetadataiTunesMetadataKeyArtDirector;
NSString *const AVMetadataiTunesMetadataKeyArranger;
NSString *const AVMetadataiTunesMetadataKeyAuthor;
NSString *const AVMetadataiTunesMetadataKeyLyrics;
NSString *const AVMetadataiTunesMetadataKeyAcknowledgement;
NSString *const AVMetadataiTunesMetadataKeyConductor;
NSString *const AVMetadataiTunesMetadataKeyDescription;
NSString *const AVMetadataiTunesMetadataKeyDirector;
NSString *const AVMetadataiTunesMetadataKeyEQ;
NSString *const AVMetadataiTunesMetadataKeyLinerNotes;
NSString *const AVMetadataiTunesMetadataKeyRecordCompany;
NSString *const AVMetadataiTunesMetadataKeyOriginalArtist;
NSString *const AVMetadataiTunesMetadataKeyPhonogramRights;
NSString *const AVMetadataiTunesMetadataKeyProducer;
NSString *const AVMetadataiTunesMetadataKeyPerformer;
NSString *const AVMetadataiTunesMetadataKeyPublisher;
NSString *const AVMetadataiTunesMetadataKeySoundEngineer;
NSString *const AVMetadataiTunesMetadataKeySoloist;
NSString *const AVMetadataiTunesMetadataKeyCredits;
NSString *const AVMetadataiTunesMetadataKeyThanks;
NSString *const AVMetadataiTunesMetadataKeyOnlineExtras;
NSString *const AVMetadataiTunesMetadataKeyExecProducer;
```

Constants

`AVMetadataiTunesMetadataKeyAlbum`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataiTunesMetadataKeyArtist`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataiTunesMetadataKeyUserComment`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataiTunesMetadataKeyCoverArt`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataiTunesMetadataKeyCopyright`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataiTunesMetadataKeyReleaseDate`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataiTunesMetadataKeyEncodedBy`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataiTunesMetadataKeyPredefinedGenre`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataiTunesMetadataKeyUserGenre`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataiTunesMetadataKeySongName`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataiTunesMetadataKeyTrackSubTitle`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataiTunesMetadataKeyEncodingTool`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataiTunesMetadataKeyComposer`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataiTunesMetadataKeyAlbumArtist`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataiTunesMetadataKeyAccountKind`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataiTunesMetadataKeyAppleID`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataiTunesMetadataKeyArtistID`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataiTunesMetadataKeySongID`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataiTunesMetadataKeyDiscCompilation`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataiTunesMetadataKeyDiscNumber`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataiTunesMetadataKeyGenreID`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataiTunesMetadataKeyGrouping`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataiTunesMetadataKeyPlaylistID`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataiTunesMetadataKeyContentRating`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataiTunesMetadataKeyBeatsPerMin`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataiTunesMetadataKeyTrackNumber`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataiTunesMetadataKeyArtDirector`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataiTunesMetadataKeyArranger`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataiTunesMetadataKeyAuthor`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataiTunesMetadataKeyLyrics`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataiTunesMetadataKeyAcknowledgement`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataiTunesMetadataKeyConductor`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataiTunesMetadataKeyDescription`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataiTunesMetadataKeyDirector`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataiTunesMetadataKeyEQ`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataiTunesMetadataKeyLinerNotes`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataiTunesMetadataKeyRecordCompany`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataiTunesMetadataKeyOriginalArtist`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataiTunesMetadataKeyPhonogramRights`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataiTunesMetadataKeyProducer`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataiTunesMetadataKeyPerformer`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataiTunesMetadataKeyPublisher`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataiTunesMetadataKeySoundEngineer`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataiTunesMetadataKeySoloist`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataiTunesMetadataKeyCredits`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataiTunesMetadataKeyThanks`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataiTunesMetadataKeyOnlineExtras`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataiTunesMetadataKeyExecProducer`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

AV Foundation QuickTime Constants

Framework: AVFoundation/AVFoundation.h

Declared in AVMetadataFormat.h

Overview

This document describes constants defined in the AV Foundation framework related to QuickTime.

Constants

QuickTime User Data

```
NSString *const AVMetadataFormatQuickTimeUserData;
NSString *const AVMetadataKeySpaceQuickTimeUserData;
```

Constants

`AVMetadataFormatQuickTimeUserData`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataKeySpaceQuickTimeUserData`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

QuickTime User Data Keys

QuickTime user data keys.

```
NSString *const AVMetadataQuickTimeUserDataKeyAlbum;
NSString *const AVMetadataQuickTimeUserDataKeyArranger;
NSString *const AVMetadataQuickTimeUserDataKeyArtist;
NSString *const AVMetadataQuickTimeUserDataKeyAuthor;
NSString *const AVMetadataQuickTimeUserDataKeyChapter;
NSString *const AVMetadataQuickTimeUserDataKeyComment;
NSString *const AVMetadataQuickTimeUserDataKeyComposer;
NSString *const AVMetadataQuickTimeUserDataKeyCopyright;
NSString *const AVMetadataQuickTimeUserDataKeyCreationDate;
NSString *const AVMetadataQuickTimeUserDataKeyDescription;
NSString *const AVMetadataQuickTimeUserDataKeyDirector;
NSString *const AVMetadataQuickTimeUserDataKeyDisclaimer;
NSString *const AVMetadataQuickTimeUserDataKeyEncodedBy;
NSString *const AVMetadataQuickTimeUserDataKeyFullName;
NSString *const AVMetadataQuickTimeUserDataKeyGenre;
NSString *const AVMetadataQuickTimeUserDataKeyHostComputer;
NSString *const AVMetadataQuickTimeUserDataKeyInformation;
NSString *const AVMetadataQuickTimeUserDataKeyKeywords;
NSString *const AVMetadataQuickTimeUserDataKeyMake;
NSString *const AVMetadataQuickTimeUserDataKeyModel;
NSString *const AVMetadataQuickTimeUserDataKeyOriginalArtist;
NSString *const AVMetadataQuickTimeUserDataKeyOriginalFormat;
NSString *const AVMetadataQuickTimeUserDataKeyOriginalSource;
NSString *const AVMetadataQuickTimeUserDataKeyPerformers;
NSString *const AVMetadataQuickTimeUserDataKeyProducer;
NSString *const AVMetadataQuickTimeUserDataKeyPublisher;
NSString *const AVMetadataQuickTimeUserDataKeyProduct;
NSString *const AVMetadataQuickTimeUserDataKeySoftware;
NSString *const AVMetadataQuickTimeUserDataKeySpecialPlaybackRequirements;
NSString *const AVMetadataQuickTimeUserDataKeyTrack;
NSString *const AVMetadataQuickTimeUserDataKeyWarning;
NSString *const AVMetadataQuickTimeUserDataKeyWriter;
NSString *const AVMetadataQuickTimeUserDataKeyURLLink;
NSString *const AVMetadataQuickTimeUserDataKeyLocationISO6709;
NSString *const AVMetadataQuickTimeUserDataKeyTrackName;
NSString *const AVMetadataQuickTimeUserDataKeyCredits;
NSString *const AVMetadataQuickTimeUserDataKeyPhonogramRights;
NSString *const AVMetadataQuickTimeMetadataKeyCameraIdentifier;
NSString *const AVMetadataQuickTimeMetadataKeyCameraFrameReadoutTime;

NSString *const AVMetadataISOUserDataKeyCopyright;
NSString *const AVMetadata3GPUserDataKeyCopyright;
NSString *const AVMetadata3GPUserDataKeyAuthor;
NSString *const AVMetadata3GPUserDataKeyPerformer;
NSString *const AVMetadata3GPUserDataKeyGenre;
NSString *const AVMetadata3GPUserDataKeyRecordingYear;
NSString *const AVMetadata3GPUserDataKeyLocation;
NSString *const AVMetadata3GPUserDataKeyTitle;
NSString *const AVMetadata3GPUserDataKeyDescription;
```

Constants

`AVMetadataQuickTimeUserDataKeyAlbum`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeUserDataKeyArranger`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeUserDataKeyArtist`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeUserDataKeyAuthor`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeUserDataKeyChapter`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeUserDataKeyComment`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeUserDataKeyComposer`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeUserDataKeyCopyright`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeUserDataKeyCreationDate`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeUserDataKeyDescription`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeUserDataKeyDirector`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeUserDataKeyDisclaimer`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeUserDataKeyEncodedBy`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeUserDataKeyFullName`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeUserDataKeyGenre`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeUserDataKeyHostComputer`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeUserDataKeyInformation`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeUserDataKeyKeywords`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeUserDataKeyMake`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeUserDataKeyModel`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeUserDataKeyOriginalArtist`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeUserDataKeyOriginalFormat`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeUserDataKeyOriginalSource`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeUserDataKeyPerformers`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeUserDataKeyProducer`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeUserDataKeyPublisher`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeUserDataKeyProduct`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeUserDataKeySoftware`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeUserDataKeySpecialPlaybackRequirements`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeUserDataKeyTrack`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeUserDataKeyWarning`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeUserDataKeyWriter`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeUserDataKeyURLLink`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeUserDataKeyLocationISO6709`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeUserDataKeyTrackName`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeUserDataKeyCredits`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeUserDataKeyPhonogramRights`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeMetadataKeyCameraIdentifier`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeMetadataKeyCameraFrameReadoutTime`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataISOUserDataKeyCopyright`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadata3GPUserDataKeyCopyright`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadata3GPUserDataKeyAuthor`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadata3GPUserDataKeyPerformer`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadata3GPUserDataKeyGenre`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadata3GPUserDataKeyRecordingYear`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadata3GPUserDataKeyLocation`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadata3GPUserDataKeyTitle`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadata3GPUserDataKeyDescription`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

QuickTime Metadata

QuickTime metadata.

```
NSString *const AVMetadataFormatQuickTimeMetadata;
NSString *const AVMetadataKeySpaceQuickTimeMetadata;
```

Constants

`AVMetadataFormatQuickTimeMetadata`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataKeySpaceQuickTimeMetadata`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

QuickTime Metadata Keys

QuickTime metadata keys.

```
NSString *const AVMetadataQuickTimeMetadataKeyAuthor;
NSString *const AVMetadataQuickTimeMetadataKeyComment;
NSString *const AVMetadataQuickTimeMetadataKeyCopyright;
NSString *const AVMetadataQuickTimeMetadataKeyCreationDate;
NSString *const AVMetadataQuickTimeMetadataKeyDirector;
NSString *const AVMetadataQuickTimeMetadataKeyDisplayName;
NSString *const AVMetadataQuickTimeMetadataKeyInformation;
NSString *const AVMetadataQuickTimeMetadataKeyKeywords;
NSString *const AVMetadataQuickTimeMetadataKeyProducer;
NSString *const AVMetadataQuickTimeMetadataKeyPublisher;
NSString *const AVMetadataQuickTimeMetadataKeyAlbum;
NSString *const AVMetadataQuickTimeMetadataKeyArtist;
NSString *const AVMetadataQuickTimeMetadataKeyArtwork;
NSString *const AVMetadataQuickTimeMetadataKeyDescription;
NSString *const AVMetadataQuickTimeMetadataKeySoftware;
NSString *const AVMetadataQuickTimeMetadataKeyYear;
NSString *const AVMetadataQuickTimeMetadataKeyGenre;
NSString *const AVMetadataQuickTimeMetadataKeyiXML;
NSString *const AVMetadataQuickTimeMetadataKeyLocationISO6709;
NSString *const AVMetadataQuickTimeMetadataKeyMake;
NSString *const AVMetadataQuickTimeMetadataKeyModel;
NSString *const AVMetadataQuickTimeMetadataKeyArranger;
NSString *const AVMetadataQuickTimeMetadataKeyEncodedBy;
NSString *const AVMetadataQuickTimeMetadataKeyOriginalArtist;
NSString *const AVMetadataQuickTimeMetadataKeyPerformer;
NSString *const AVMetadataQuickTimeMetadataKeyComposer;
NSString *const AVMetadataQuickTimeMetadataKeyCredits;
NSString *const AVMetadataQuickTimeMetadataKeyPhonogramRights;
```

Constants

`AVMetadataQuickTimeMetadataKeyAuthor`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeMetadataKeyComment`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeMetadataKeyCopyright`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeMetadataKeyCreationDate`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeMetadataKeyDirector`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeMetadataKeyDisplayName`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeMetadataKeyInformation`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeMetadataKeyKeywords`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeMetadataKeyProducer`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeMetadataKeyPublisher`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeMetadataKeyAlbum`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeMetadataKeyArtist`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeMetadataKeyArtwork`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeMetadataKeyDescription`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeMetadataKeySoftware`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeMetadataKeyYear`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeMetadataKeyGenre`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeMetadataKeyiXML`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeMetadataKeyLocationISO6709`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeMetadataKeyMake`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeMetadataKeyModel`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeMetadataKeyArranger`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeMetadataKeyEncodedBy`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeMetadataKeyOriginalArtist`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeMetadataKeyPerformer`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeMetadataKeyComposer`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeMetadataKeyCredits`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeMetadataKeyPhonogramRights`

Available in iOS 4.0 and later.

Declared in `AVMetadataFormat.h`.

Document Revision History

This table describes the changes to *AV Foundation Framework Reference*.

Date	Notes
2010-07-13	Corrected minor typographical error.
2010-05-15	Updated for iPhone OS 4.0.
2009-03-02	Updated for iPhone OS 3.0
	Added classes for audio recording and audio session management.
2008-11-07	New document that describes the interfaces in the AV Foundation framework.

Document Revision History