# AV Foundation Framework Reference

**Audio & Video**

**2011-01-06**

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

# Contents

# AV Foundation Framework Reference

| | |
|---|---|
| **Framework** | /System/Library/Frameworks/AVFoundation.framework |
| **Header file directories** | /System/Library/Frameworks/AVFoundation.framework/Headers |
| **Declared in** | AVAnimation.h<br>AVAsset.h<br>AVAssetExportSession.h<br>AVAssetImageGenerator.h<br>AVAssetReader.h<br>AVAssetReaderOutput.h<br>AVAssetTrack.h<br>AVAssetTrackSegment.h<br>AVAssetWriter.h<br>AVAssetWriterInput.h<br>AVAsynchronousKeyValueLoading.h<br>AVAudioMix.h<br>AVAudioPlayer.h<br>AVAudioRecorder.h<br>AVAudioSettings.h<br>AVCaptureDevice.h<br>AVCaptureInput.h<br>AVCaptureOutput.h<br>AVCaptureSession.h<br>AVCaptureVideoPreviewLayer.h<br>AVComposition.h<br>AVCompositionTrack.h<br>AVCompositionTrackSegment.h<br>AVError.h<br>AVMediaFormat.h<br>AVMetadataFormat.h<br>AVMetadataItem.h<br>AVPlayer.h<br>AVPlayerItem.h<br>AVPlayerItemTrack.h<br>AVPlayerLayer.h<br>AVSynchronizedLayer.h<br>AVTime.h<br>AVTimedMetadataGroup.h<br>AVUtilities.h<br>AVVideoComposition.h<br>AVVideoSettings.h |

# Introduction

The AV Foundation framework provides an Objective-C interface for managing and playing audio-visual media in your Mac OS X application. To learn more about AV Foundation, see *AV Foundation Programming Guide*.

# Classes

# AVAsset Class Reference

| | |
|---|---|
| **Inherits from** | NSObject |
| **Conforms to** | NSCopying<br>AVAsynchronousKeyValueLoading<br>NSObject (NSObject) |
| **Framework** | /System/Library/Frameworks/AVFoundation.framework |
| **Availability** | Available in iOS 4.0 and later. |
| **Declared in** | AVAsset.h<br>AVVideoComposition.h |
| **Companion guide** | AV Foundation Programming Guide |

## 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`.

### 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 27)
  Cancels the loading of all values for all observers.

## Accessing Metadata

`commonMetadata` (page 22) *property*
An array of metadata items for each common metadata key for which a value is available. (read-only)

`availableMetadataFormats` (page 22) *property*
An array of strings, each representing a metadata format that's available to the asset. (read-only)

- `metadataForFormat:` (page 28)
  Returns an array of `AVMetadataItem` objects, one for each metadata item in the container of the specified format

`lyrics` (page 24) *property*
The lyrics of the asset suitable for the current locale. (read-only)

`availableChapterLocales` (page 22) *property*
The locales available for chapters in the asset. (read-only)

- `chapterMetadataGroupsWithTitleLocale:containingItemsWithCommonKeys:` (page 27)
  Returns an array of chapters with a given title locale and containing specified keys.

## Accessing Tracks

`tracks` (page 26) *property*
The tracks contained by the asset. (read-only)

- `trackWithTrackID:` (page 29)
  Returns the track with a specified track ID.
- `tracksWithMediaCharacteristic:` (page 28)
  Returns an array of `AVAssetTrack` objects of the asset that present media with a specified characteristic.
- `tracksWithMediaType:` (page 29)
  Returns an array of the asset tracks of the asset that present media of a specified type.

## Determining Usability

`hasProtectedContent` (page 24) *property*
Indicates whether the asset has protected content. (read-only)

`playable` (page 24) *property*
Indicates whether the asset, or its URL, can be used to initialize an instance of `AVPlayerItem`. (read-only)

`exportable` (page 23) *property*
Indicates whether the asset can be exported using `AVAssetExportSession`. (read-only)

`readable` (page 26) *property*
Indicates whether the asset's media data can be extracted using `AVAssetReader`. (read-only)

`composable` (page 23) *property*
Indicates whether the asset can be used within a segment of an `AVCompositionTrack` object. (read-only)

## AVAssetVideoCompositionUtility

- `unusedTrackID` (page 30)
Returns a track ID for the asset.

## Accessing Common Metadata

`duration` (page 23) *property*
The duration of the asset. (read-only)

`providesPreciseDurationAndTiming` (page 25) *property*
Indicates whether the asset provides precise timing. (read-only)

## Preferred Asset Attributes

`naturalSize` (page 24) *property*
The encoded or authored size of the visual portion of the asset. (read-only)

`preferredRate` (page 25) *property*
The natural rate at which the asset is to be played. (read-only)

`preferredTransform` (page 25) *property*
The preferred transform to apply to the visual content of the asset for presentation or processing. (read-only)

`preferredVolume` (page 25) *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*.

## availableChapterLocales

The locales available for chapters in the asset. (read-only)

```
@property(readonly) NSArray *availableChapterLocales
```

**Discussion**
The array contains instances of `NSLocale`.

**Availability**
Available in iOS 4.3 and later.

**See Also**
`- chapterMetadataGroupsWithTitleLocale:containingItemsWithCommonKeys:` (page 27)

**Declared In**
`AVAsset.h`

## 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 237) (`AVMetadataItem`) or by key using `metadataItemsFromArray:withKey:keySpace:` (page 236) (`AVMetadataItem`).

**Availability**
Available in iOS 4.0 and later.

**Declared In**
`AVAsset.h`

## composable

Indicates whether the asset can be used within a segment of an `AVCompositionTrack` object. (read-only)

```
@property(nonatomic, readonly, getter=isComposable) BOOL composable
```

**Availability**
Available in iOS 4.3 and later.

**Declared In**
`AVAsset.h`

## duration

The duration of the asset. (read-only)

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

**Discussion**
If `providesPreciseDurationAndTiming` (page 25) 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`

## exportable

Indicates whether the asset can be exported using `AVAssetExportSession`. (read-only)

```
@property(nonatomic, readonly, getter=isExportable) BOOL exportable
```

**Availability**
Available in iOS 4.3 and later.

**Declared In**
`AVAsset.h`

## hasProtectedContent

Indicates whether the asset has protected content. (read-only)

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

**Availability**
Available in iOS 4.2 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`

## playable

Indicates whether the asset, or its URL, can be used to initialize an instance of `AVPlayerItem`. (read-only)

```
@property(nonatomic, readonly, getter=isPlayable) BOOL playable
```

**Availability**
Available in iOS 4.3 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 23)

**Declared In**
`AVAsset.h`

## readable

Indicates whether the asset's media data can be extracted using `AVAssetReader`. (read-only)

```
@property(nonatomic, readonly, getter=isReadable) BOOL readable
```

**Availability**
Available in iOS 4.3 and later.

**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 29)
`- tracksWithMediaCharacteristic:` (page 28)
`- trackWithTrackID:` (page 29)

**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`

## chapterMetadataGroupsWithTitleLocale:containingItemsWithCommonKeys:

Returns an array of chapters with a given title locale and containing specified keys.

```
- (NSArray *)chapterMetadataGroupsWithTitleLocale:(NSLocale *)locale
    containingItemsWithCommonKeys:(NSArray *)commonKeys
```

**Parameters**

*`locale`*
The locale of the metadata items carrying chapter titles to be returned (the method supports the IETF BCP 47 specification of locales).

*`commonKeys`*
An array of common keys of `AVMetadataItem` to include in the returned array. `AVMetadataCommonKeyArtwork` is the only supported key.

**Return Value**
An array of `AVTimedMetadataGroup` objects.

**Discussion**
Each object in the returned array contains an `AVMetadataItem` object representing the chapter title, and the time range property of the `AVTimedMetadataGroup` object is equal to the time range of the chapter title item.

An `AVMetadataItem` with the specified common key is added to an existing `AVTimedMetadataGroup` object if the time range (timestamp and duration) of the metadata item and the metadata group overlap.

The locale of items not carrying chapter titles need not match the specified locale parameter. You can filter the returned items based on locale using `metadataItemsFromArray:withLocale:`.

**Availability**
Available in iOS 4.3 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 237) (`AVMetadataItem`) or by key using `metadataItemsFromArray:withKey:keySpace:` (page 236) (`AVMetadataItem`).

**Special Considerations**

Becomes callable without blocking when `availableMetadataFormats` (page 22) 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 26) has been loaded.

**Availability**

Available in iOS 4.0 and later.

**See Also**

- `tracksWithMediaType:` (page 29)
- `trackWithTrackID:` (page 29)

`@property tracks` (page 26)

**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 26) has been loaded.

**Availability**
Available in iOS 4.0 and later.

**See Also**
- tracksWithMediaCharacteristic: (page 28)
- trackWithTrackID: (page 29)

  @property tracks (page 26)

**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 26) has been loaded.

**Availability**
Available in iOS 4.0 and later.

**See Also**
- tracksWithMediaType: (page 29)
- tracksWithMediaCharacteristic: (page 28)

`@property tracks` (page 26)

**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 |
| **Companion guide** | AV Foundation Programming Guide |

## 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.

Prior to initializing an instance of `AVAssetExportSession`, you can use `allExportPresets` (page 38) to get the complete list of presets available. Use `exportPresetsCompatibleWithAsset:` (page 39) to get a list of presets that are compatible with a specific asset.

After you have initialized an export session with the asset that contains the source media, the export preset name, and the output file type (a UTI string from among those defined in `AVMediaFormat.h`), you can start the export running by invoking `exportAsynchronouslyWithCompletionHandler:` (page 40). Because the export is performed asynchronously, this method returns immediately—you can invoke use `progress` (page 36) to check on the progress. Note that in some cases, depending on the capabilities of the device, when multiple exports are attempted at the same time, some may be queued until others have been completed. When this happens, the `status` (page 37) of a queued export will indicate that it's "waiting" (`AVAssetExportSessionStatusWaiting` (page 42)).

The completion handler you supply to `exportAsynchronouslyWithCompletionHandler:` (page 40) is called whether the export fails, completes, or is cancelled. Upon completion, the `status` (page 37) property indicates whether the export completed successfully. If it failed, the value of the `error` (page 34) property gives additional information about the reason.

# Tasks

## Initializing a Session

- `initWithAsset:presetName:` (page 41)
  Initializes an asset export session with a specified asset and preset.

+ `exportSessionWithAsset:presetName:` (page 39)
  Returns an asset export session configured with a specified asset and preset.

## Exporting

- `exportAsynchronouslyWithCompletionHandler:` (page 40)
  Starts the asynchronous execution of an export session.

- `cancelExport` (page 40)
  Cancels the execution of an export session.

`error` (page 34) *property*
  Describes the error that occurred if the export status is `AVAssetExportSessionStatusFailed` or `AVAssetExportSessionStatusCancelled`. (read-only)

`maxDuration` (page 34) *property*
  The maximum duration that is allowed for export. (read-only)

## Export Status

`progress` (page 36) *property*
  The progress of the export on a scale from 0 to 1. (read-only)

`status` (page 37) *property*
  The status of the export session. (read-only)

## Configuring Output

`outputURL` (page 35) *property*
  The URL of the export session's output.

`supportedFileTypes` (page 37) *property*
  The types of files the session can write. (read-only)

`outputFileType` (page 35) *property*
The type of file to be written by the session.

`fileLengthLimit` (page 34) *property*
The maximum number of bytes that the session is allowed to write to the output URL.

`timeRange` (page 38) *property*
The time range to be exported from the source.

`metadata` (page 35) *property*
The metadata to be written to the output file by the export session.

`audioMix` (page 33) *property*
Indicates whether non-default audio mixing is enabled for export, and supplies the parameters for audio mixing.

`shouldOptimizeForNetworkUse` (page 37) *property*
Indicates whether the movie should be optimized for network use.

`videoComposition` (page 38) *property*
Indicates whether video composition is enabled for export, and supplies the instructions for video composition.

---

### Export Presets

`presetName` (page 36) *property*
The name of the preset with which the session was initialized. (read-only)

`+ allExportPresets` (page 38)
Returns all available export preset names.

`+ exportPresetsCompatibleWithAsset:` (page 39)
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 40)
  `@property status` (page 37)

**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 38).

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 37)
`@property outputURL` (page 35)

**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 35), `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 35)

**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 43), "Export Preset Names for QuickTime Files of a Given Size" (page 43), AVAssetExportSessionStatusCancelled (page 42), "Export Preset Name for iTunes Audio" (page 44), and "Export Preset Name for Pass-Through" (page 44).

You can observe this property using key-value observing.

**Availability**
Available in iOS 4.0 and later.

**See Also**
`- initWithAsset:presetName:` (page 39)

**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 "AVAssetExportSessionStatus" (page 42).

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 35)

**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 43), "Export Preset Names for QuickTime Files of a Given Size" (page 43), AVAssetExportSessionStatusCancelled (page 42), "Export Preset Name for iTunes Audio" (page 44), and "Export Preset Name for Pass-Through" (page 44).

**Discussion**
Not all presets are compatible with all assets.

**Availability**
Available in iOS 4.0 and later.

**See Also**
`+ exportPresetsCompatibleWithAsset:` (page 39)

**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 39) with the specified asset. For possible values, see "Export Preset Names for Device-Appropriate QuickTime Files" (page 43), "Export Preset Names for QuickTime Files of a Given Size" (page 43), AVAssetExportSessionStatusCancelled (page 42), "Export Preset Name for iTunes Audio" (page 44), and "Export Preset Name for Pass-Through" (page 44).

**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 38)

**Declared In**
`AVAssetExportSession.h`

## exportSessionWithAsset:presetName:

Returns an asset export session configured with a specified asset and preset.

```
+ (id)exportSessionWithAsset:(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 43), "Export Preset Names for QuickTime Files of a Given Size" (page 43), AVAssetExportSessionStatusCancelled (page 42), "Export Preset Name for iTunes Audio" (page 44), and "Export Preset Name for Pass-Through" (page 44).

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

**Availability**
Available in iOS 4.1 and later.

**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 37) signals the terminal state of the export session, and if a failure occurs, `error` (page 34) 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 40) is invoked.
3. After the export session succeeds, having completely written its output to the `outputURL` (page 35).

**Availability**
Available in iOS 4.0 and later.

**See Also**
- `cancelExport` (page 40)
`@property status` (page 37)
`@property error` (page 34)

**Declared In**
`AVAssetExportSession.h`

## initWithAsset:presetName:

Initializes an asset export session with a specified asset and preset.

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

**Parameters**

*asset*
The asset 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 43), "Export Preset Names for QuickTime Files of a Given Size" (page 43), AVAssetExportSessionStatusCancelled (page 42), "Export Preset Name for iTunes Audio" (page 44), and "Export Preset Name for Pass-Through" (page 44).

**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

Constants to indicate the status of the session.

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

**Constants**

`AVAssetExportSessionStatusUnknown`
Indicates that the status is unknown.
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`.

`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`.

## 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 42).

## 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;
```

**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`.

**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 38) and `exportPresetsCompatibleWithAsset:` (page 39) 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 49), which employs decoding efficiencies similar to those used during playback.

You create an asset generator using `initWithAsset:` (page 50) or `assetImageGeneratorWithAsset:` (page 48). 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 28) (`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 50)
Initializes an image generator for use with a specified asset.

`+ assetImageGeneratorWithAsset:` (page 48)
Returns an image generator for use with a specified asset.

### Generating Images

- `copyCGImageAtTime:actualTime:error:` (page 49)
  Returns a CGImage for the asset at or near a specified time.
- `generateCGImagesAsynchronouslyForTimes:completionHandler:` (page 49)
  Creates a series of CGImage objects for an asset at or near specified times.
- `cancelAllCGImageGeneration` (page 48)
  Cancels all pending image generation requests.

### Generation Behavior

`apertureMode` (page 46) *property*
Specifies the aperture mode for the generated image.

`appliesPreferredTrackTransform` (page 47) *property*
Specifies whether to apply the track matrix (or matrices) when extracting an image from the asset.

`maximumSize` (page 47) *property*
Specifies the maximum dimensions for generated image.

`videoComposition` (page 47) *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 51).

**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 47) property.

**Availability**
Available in iOS 4.0 and later.

**See Also**
`preferredTransform`
`@property videoComposition` (page 47)

**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 46) 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 47) property is ignored.

**Availability**
Available in iOS 4.0 and later.

**See Also**
`@property appliesPreferredTrackTransform` (page 47)

**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 28)

**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 52) for each image time in every previous invocation of generateCGImagesAsynchronouslyForTimes:completionHandler: (page 49) for which images have not yet been supplied.

**Availability**
Available in iOS 4.0 and later.

**See Also**
- copyCGImageAtTime:actualTime:error: (page 49)

- generateCGImagesAsynchronouslyForTimes:completionHandler: (page 49)

**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 49)

**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 48)
- `copyCGImageAtTime:actualTime:error:` (page 49)

**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 28)

**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 49) 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 52), 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 49).

# AVAssetReader Class Reference

| | |
|---|---|
| **Inherits from** | NSObject |
| **Conforms to** | NSObject (NSObject) |
| **Framework** | /System/Library/Frameworks/AVFoundation.framework |
| **Availability** | Available in iOS 4.1 and later. |
| **Declared in** | AVAssetReader.h |
| **Companion guide** | AV Foundation Programming Guide |

## Overview

You use an `AVAssetReader` object to obtaining media data of an asset, whether the asset is file-based or represents an assemblage of media data from multiple sources (as with an `AVComposition` object).

`AVAssetReader` lets you:

- Read raw un-decoded media samples directly from storage, obtain samples decoded into renderable forms.
- Mix multiple audio tracks of the asset and compose multiple video tracks (by using `AVAssetReaderAudioMixOutput` and `AVAssetReaderVideoCompositionOutput`).

`AVAssetReader`'s pipelines are multithreaded internally. After you initiate reading with `initWithAsset:error:` (page 58), a reader loads and processes a reasonable amount of sample data ahead of use so that retrieval operations such as `copyNextSampleBuffer` (page 66) (`AVAssetReaderOutput`) can have very low latency. Note, however, that `AVAssetReader` is not intended for use with real-time sources, and its performance is not guaranteed for real-time operations.

## Tasks

### Creating a Reader

`- initWithAsset:error:` (page 58)
Initializes an asset reader for reading media data from a specified asset.

+ `assetReaderWithAsset:error:` (page 56)
Returns an asset reader for reading media data from a specified asset.

---

### Managing Outputs

`outputs` (page 55) *property*
The outputs from which clients of reader can read media data. (read-only)

- `addOutput:` (page 57)
Adds a given output to the receiver.

- `canAddOutput:` (page 58)
Returns a Boolean value that indicates whether a given output can be added to the receiver.

---

### Controlling Reading

`status` (page 56) *property*
The status of the reading of sample buffers from the asset. (read-only)

- `startReading` (page 59)
Prepares the receiver for obtaining sample buffers from the asset.

- `cancelReading` (page 58)
Cancels any background work and prevents the receiver's outputs from reading more samples.

`error` (page 55) *property*
Describes the error that occurred if the status is `AVAssetReaderStatusFailed`. (read-only)

`timeRange` (page 56) *property*
The time range of the asset that should be read.

---

### Asset Properties

`asset` (page 55) *property*
The asset with which the receiver was initialized. (read-only)

## Properties

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

## asset

The asset with which the receiver was initialized. (read-only)

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

**Discussion**
Concrete instances of `AVAssetReader` with specific `AVAssetTrack` instances must obtain those tracks from the asset returned by this property.

**Availability**
Available in iOS 4.1 and later.

**Declared In**
`AVAssetReader.h`

## error

Describes the error that occurred if the status is `AVAssetReaderStatusFailed`. (read-only)

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

**Discussion**
This property is thread safe.

The value of this property describes what caused the reader to no longer be able to read its asset. If the reader's status is not `AVAssetReaderStatusFailed` (page 60), the value of this property is `nil`.

**Availability**
Available in iOS 4.1 and later.

**See Also**
- `startReading` (page 59)
- `@property status` (page 56)

**Declared In**
`AVAssetReader.h`

## outputs

The outputs from which clients of reader can read media data. (read-only)

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

**Discussion**
The array contains concrete instances of `AVAssetReaderOutput` associated with the reader.

**Availability**
Available in iOS 4.1 and later.

**See Also**
- `canAddOutput:` (page 58)
- `addOutput:` (page 57)

**Declared In**
`AVAssetReader.h`

### status

The status of the reading of sample buffers from the asset. (read-only)

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

**Discussion**
This property is thread safe. For possible values, see "Reader Status Constants" (page 60).

The value of this property indicates whether reading is in progress, has completed successfully, has been canceled, or has failed. You should check the value of this property `copyNextSampleBuffer` (page 66) (`AVAssetReaderOutput`) returns `NULL` to determine why no more samples could be read. */

**Availability**
Available in iOS 4.1 and later.

**Declared In**
`AVAssetReader.h`

### timeRange

The time range of the asset that should be read.

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

**Discussion**
The intersection of the value of this property and `CMTimeRangeMake(kCMTimeZero, asset.duration)` determines the time range of the asset from which media data will be read.

The default value is `CMTimeRangeMake(kCMTimeZero, kCMTimePositiveInfinity)`. You cannot change the value of this property after reading has started.

**Availability**
Available in iOS 4.1 and later.

**Declared In**
`AVAssetReader.h`

## Class Methods

### assetReaderWithAsset:error:

Returns an asset reader for reading media data from a specified asset.

```
+ (AVAssetReader *)assetReaderWithAsset:(AVAsset *)asset error:(NSError **)outError
```

**Parameters**

*asset*

The asset from which media data is to be read.

*outError*

If initialization of the reader fails, upon return contains an error that describes the problem.

**Return Value**

An asset reader, initialized for reading media data from *asset*.

**Availability**

Available in iOS 4.1 and later.

**See Also**

- initWithAsset:error: (page 58)

**Declared In**

AVAssetReader.h

# Instance Methods

## addOutput:

Adds a given output to the receiver.

```
- (void)addOutput:(AVAssetReaderOutput *)output
```

**Parameters**

*output*

The reader output to add.

**Discussion**

Outputs are created with a reference to one or more AVAssetTrack objects. Adding an output to an asset reader indicates to the reader that it should source from those tracks. The tracks must be owned by the asset returned by the reader's asset property.

You cannot add an output after reading has started.

**Availability**

Available in iOS 4.1 and later.

**See Also**

- canAddOutput: (page 58)
- @property outputs (page 55)
- @property asset (page 55)

**Declared In**

AVAssetReader.h

## canAddOutput:

Returns a Boolean value that indicates whether a given output can be added to the receiver.

```
- (BOOL)canAddOutput:(AVAssetReaderOutput *)output
```

**Parameters**

*output*

The reader output to be tested.

**Return Value**

`YES` if *output* can be added to the receiver, otherwise `NO`.

**Discussion**

You cannot add an output that reads from a track of an asset other than the asset used to initialize the receiver.

**Availability**

Available in iOS 4.1 and later.

**See Also**

- `addOutput:` (page 57)
- `@property outputs` (page 55)

**Declared In**

`AVAssetReader.h`

## cancelReading

Cancels any background work and prevents the receiver's outputs from reading more samples.

```
- (void)cancelReading
```

**Discussion**

If you want to stop reading samples from the receiver before reaching the end of its time range, you should call this method to stop any background read ahead operations that the may have been in progress.

**Availability**

Available in iOS 4.1 and later.

**See Also**

- `startReading` (page 59)
- `@property status` (page 56)
- `@property error` (page 55)

**Declared In**

`AVAssetReader.h`

## initWithAsset:error:

Initializes an asset reader for reading media data from a specified asset.

```
- (id)initWithAsset:(AVAsset *)asset error:(NSError **)outError
```

**Parameters**

*asset*

The asset from which media data is to be read.

*outError*

If initialization of the reader fails, upon return contains an error that describes the problem.

**Return Value**
An asset reader, initialized for reading media data from *asset*.

**Availability**
Available in iOS 4.1 and later.

**See Also**
`+ assetReaderWithAsset:error:` (page 56)

**Declared In**
`AVAssetReader.h`

### startReading

Prepares the receiver for obtaining sample buffers from the asset.

```
- (BOOL)startReading
```

**Return Value**
`YES` if the reader is able to start reading, otherwise `NO`.

**Discussion**
This method validates the entire collection of settings for outputs for tracks, for audio mixdown, and for video composition and initiates reading of all outputs.

`status` (page 56) signals the terminal state of the asset reader, and if a failure occurs, `error` (page 55) describes the failure.

**Availability**
Available in iOS 4.1 and later.

**See Also**
`- cancelReading` (page 58)
`@property status` (page 56)
`@property error` (page 55)

**Declared In**
`AVAssetReader.h`

## Constants

### AVAssetReaderStatus

A type for constants to indicate the reader's status.

```
typedef NSInteger AVAssetReaderStatus;
```

**Discussion**
For possible values, see "Reader Status Constants" (page 60).

**Availability**
Available in iOS 4.1 and later.

**Declared In**
`AVAssetReader.h`

## Reader Status Constants

Constants that indicate the reader's status.

```
enum {
        AVAssetReaderStatusUnknown = 0,
        AVAssetReaderStatusReading,
        AVAssetReaderStatusCompleted,
        AVAssetReaderStatusFailed,
        AVAssetReaderStatusCancelled,
};
```

**Constants**

`AVAssetReaderStatusUnknown`
Indicates that `startReading` (page 59) has not yet been invoked.
Available in iOS 4.1 and later.
Declared in `AVAssetReader.h`.

`AVAssetReaderStatusReading`
Indicates that the reader is ready to provide more sample buffers to its outputs.
Available in iOS 4.1 and later.
Declared in `AVAssetReader.h`.

`AVAssetReaderStatusCompleted`
Indicates that the reader has provided all available sample buffers to all of its outputs.
Available in iOS 4.1 and later.
Declared in `AVAssetReader.h`.

`AVAssetReaderStatusFailed`
Indicates that reading failed.
Available in iOS 4.1 and later.
Declared in `AVAssetReader.h`.

`AVAssetReaderStatusCancelled`
Indicates that reading was cancelled using `cancelReading` (page 58).
Available in iOS 4.1 and later.
Declared in `AVAssetReader.h`.

**Discussion**
You access the reader's status using the `status` (page 56) property.

# AVAssetReaderAudioMixOutput Class Reference

| | |
|---|---|
| **Inherits from** | AVAssetReaderOutput : NSObject |
| **Conforms to** | NSObject (NSObject) |
| **Framework** | /System/Library/Frameworks/AVFoundation.framework |
| **Availability** | Available in iOS 4.1 and later. |
| **Declared in** | AVAssetReaderOutput.h |
| **Companion guide** | AV Foundation Programming Guide |

## Overview

`AVAssetReaderAudioMixOutput` is a concrete subclass of `AVAssetReaderOutput` that defines an interface for reading audio samples that result from mixing the audio from one or more tracks of an `AVAssetReader` object's asset.

You can read the audio data mixed from one or more asset tracks by adding an instance of `AVAssetReaderAudioMixOutput` to an asset reader using `addOutput:` (page 57). The samples can be read in a default format or can be converted to a different format.

## Tasks

### Creating an Audio Mix Output

`- initWithAudioTracks:audioSettings:` (page 64)
Initializes an instance of `AVAssetReaderAudioMixOutput` for reading mixed audio from the specified audio tracks, with optional audio settings.

`+ assetReaderAudioMixOutputWithAudioTracks:audioSettings:` (page 63)
Returns an instance of `AVAssetReaderAudioMixOutput` for reading mixed audio from the specified audio tracks, with optional audio settings.

### Settings

`audioMix` (page 62) *property*
The output's audio mix.

`audioSettings` (page 62) *property*
The audio settings used for audio output. (read-only)

`audioTracks` (page 63) *property*
The tracks from which the receiver reads mixed audio. (read-only)

## Properties

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

### audioMix

The output's audio mix.

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

**Discussion**
You use the audio mix to specify how the volume of audio samples read from each source track will change over the timeline of the source asset.

You cannot set this property after reading has started.

**Availability**
Available in iOS 4.1 and later.

**Declared In**
`AVAssetReaderOutput.h`

### audioSettings

The audio settings used for audio output. (read-only)

```
@property(nonatomic, readonly) NSDictionary *audioSettings
```

**Discussion**
The dictionary must contain values for keys in `AVAudioSettings.h` (linear PCM only).

`nil` indicates that the samples will be returned in the default format.

**Availability**
Available in iOS 4.1 and later.

**Declared In**
AVAssetReaderOutput.h

### audioTracks

The tracks from which the receiver reads mixed audio. (read-only)

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

**Discussion**
The value is an array of AVAssetTrack objects owned by the target AVAssetReader object's asset.

**Availability**
Available in iOS 4.1 and later.

**Declared In**
AVAssetReaderOutput.h

## Class Methods

### assetReaderAudioMixOutputWithAudioTracks:audioSettings:

Returns an instance of AVAssetReaderAudioMixOutput for reading mixed audio from the specified audio tracks, with optional audio settings.

```
+ (AVAssetReaderAudioMixOutput *)assetReaderAudioMixOutputWithAudioTracks:(NSArray
    *)audioTracks audioSettings:(NSDictionary *)audioSettings
```

**Parameters**

*audioTracks*

An array of AVAssetTrack objects from which the created object should read sample buffers to be mixed.

Each track must be one of the tracks owned by the target AVAssetReader object's asset and must be of media type AVMediaTypeAudio (page 400).

*audioSettings*

The audio settings to be used for audio output; the dictionary must contain values for keys in AVAudioSettings.h (linear PCM only).

Pass nil if you want to receive decoded samples in a convenient uncompressed format, with properties determined according to the properties of the specified audio tracks.

**Return Value**
An instance of AVAssetReaderAudioMixOutput for reading mixed audio from *audioTracks*, with audio settings specified by *audioSettings*.

**Discussion**
Initialization will fail if *audioSettings* cannot be used with *audioTracks*.

**Availability**
Available in iOS 4.1 and later.

**Declared In**
`AVAssetReaderOutput.h`

# Instance Methods

## initWithAudioTracks:audioSettings:

Initializes an instance of `AVAssetReaderAudioMixOutput` for reading mixed audio from the specified audio tracks, with optional audio settings.

```
- (id)initWithAudioTracks:(NSArray *)audioTracks audioSettings:(NSDictionary
    *)audioSettings
```

**Parameters**

*audioTracks*

An array of `AVAssetTrack` objects from which the created object should read sample buffers to be mixed.

Each track must be one of the tracks owned by the target `AVAssetReader` object's asset and must be of media type `AVMediaTypeAudio` (page 400).

*audioSettings*

The audio settings to be used for audio output; the dictionary must contain values for keys in `AVAudioSettings.h` (linear PCM only).

Pass `nil` if you want to receive decoded samples in a convenient uncompressed format, with properties determined according to the properties of the specified audio tracks.

**Return Value**
An instance of `AVAssetReaderAudioMixOutput` initialized for reading mixed audio from *audioTracks*, with audio settings specified by *audioSettings*.

**Discussion**
Initialization will fail if *audioSettings* cannot be used with *audioTracks*.

**Availability**
Available in iOS 4.1 and later.

**Declared In**
`AVAssetReaderOutput.h`

# AVAssetReaderOutput Class Reference

| | |
|---|---|
| **Inherits from** | NSObject |
| **Conforms to** | NSObject (NSObject) |
| **Framework** | /System/Library/Frameworks/AVFoundation.framework |
| **Availability** | Available in iOS 4.1 and later. |
| **Declared in** | AVAssetReaderOutput.h |
| **Companion guide** | AV Foundation Programming Guide |

## Overview

`AVAssetReaderOutput` is an abstract class that defines an interface for reading a single collection of samples of a common media type from an `AVAssetReader` object.

There are subclasses of `AVAssetReaderOutput` for specific tasks: `AVAssetReaderTrackOutput`, `AVAssetReaderAudioMixOutput`, and `AVAssetReaderVideoCompositionOutput`. You read the media data of an asset by adding one or more concrete instances of `AVAssetReaderOutput` to an asset reader using `addOutput:` (page 57).

## Tasks

### Copying a Buffer

`- copyNextSampleBuffer` (page 66)
Synchronously copies the next sample buffer for the output.

### Inspecting the Media Type

`mediaType` (page 66) *property*
A string representing the media type of the track (or tracks) represented by the output. (read-only)

# Properties

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

## mediaType

A string representing the media type of the track (or tracks) represented by the output. (read-only)

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

**Discussion**
The value of this property is one of the media type strings defined in AVMediaFormat.h.

**Availability**
Available in iOS 4.1 and later.

**Declared In**
`AVAssetReaderOutput.h`

# Instance Methods

## copyNextSampleBuffer

Synchronously copies the next sample buffer for the output.

```
- (CMSampleBufferRef)copyNextSampleBuffer
```

**Return Value**
The output sample buffer, or `NULL` if there are no more sample buffers available for the output within the time range specified by the asset reader's `timeRange` property. Ownership follows the "The Create Rule" in *Memory Management Programming Guide for Core Foundation*.

**Discussion**
If this method returns `NULL`, you should check the value of the associated `AVAssetReader` object's `status` (page 56) property to determine why no more samples could be read.

**Availability**
Available in iOS 4.1 and later.

**Declared In**
`AVAssetReaderOutput.h`

# AVAssetReaderTrackOutput Class Reference

| | |
|---|---|
| **Inherits from** | AVAssetReaderOutput : NSObject |
| **Conforms to** | NSObject (NSObject) |
| **Framework** | /System/Library/Frameworks/AVFoundation.framework |
| **Availability** | Available in iOS 4.1 and later. |
| **Declared in** | AVAssetReaderOutput.h |
| **Companion guide** | AV Foundation Programming Guide |

## Overview

`AVAssetReaderTrackOutput` defines an interface for reading media data from a single `AVAssetTrack` object of an asset reader's asset.

You can read the media data of an asset track by adding an instance of `AVAssetReaderTrackOutput` to an asset reader using `addOutput:` (page 57). You can read the samples in the track in the format in which they are stored in the asset, or convert them to a different format.

## Tasks

### Creating a Track Output

`+ assetReaderTrackOutputWithTrack:outputSettings:` (page 69)
Returns an asset reader wrapping a specified track, with optional output settings.

`- initWithTrack:outputSettings:` (page 69)
Initializes an asset reader to wrap a specified track, with optional output settings.

### Properties



# Properties

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

## outputSettings

The output settings used by the output. (read-only)

```
@property(nonatomic, readonly) NSDictionary *outputSettings
```

**Discussion**
The value is a dictionary that contains values for keys from either `AVAudioSettings.h` (linear PCM only) for audio tracks or `<CoreVideo/CVPixelBuffer.h>` for video tracks. A value of `nil` indicates that the output will return samples in their original format as stored in the target track.

**Availability**
Available in iOS 4.1 and later.

**Declared In**
`AVAssetReaderOutput.h`

## track

The track from which the receiver reads sample buffers. (read-only)

```
@property(nonatomic, readonly) AVAssetTrack *track
```

**Availability**
Available in iOS 4.1 and later.

**Declared In**
`AVAssetReaderOutput.h`

# Class Methods

## assetReaderTrackOutputWithTrack:outputSettings:

Returns an asset reader wrapping a specified track, with optional output settings.

```
+ (AVAssetReaderTrackOutput *)assetReaderTrackOutputWithTrack:(AVAssetTrack *)track
    outputSettings:(NSDictionary *)outputSettings
```

**Parameters**

*track*

The track from which the reader should source sample buffers.

*outputSettings*

A dictionary of output settings to be used for sample output. Pass `nil` to receive samples in their original format as stored in the track.

You use keys from one of AVAudioSettings.h (linear PCM only), AVVideoSettings.h, or <CoreVideo/CVPixelBuffer.h>, depending on the media type and the output format you want.

**Return Value**
An asset reader wrapping *track*, using the setting defined by *outputSettings*.

**Discussion**
Initialization fails if the output settings cannot be used with the specified track.

`AVAssetReaderTrackOutput` does not currently support the AVAudioSettings.h keys `AVSampleRateKey` (page 395), `AVNumberOfChannelsKey` (page 395), or `AVChannelLayoutKey` (page 397).

For optimal performance when decompressing video, the requested pixel format should match what the decoder outputs natively to avoid unnecessary conversions. For H.264 use either `kCVPixelFormatType_420YpCbCr8BiPlanarVideoRange`, or `kCVPixelFormatType_420YpCbCr8BiPlanarFullRange` if the video is known to be full range. If the pixel buffers need to be in RGB for additional processing then `kCVPixelFormatType_32BGRA` is recommended.

**Availability**
Available in iOS 4.1 and later.

**Declared In**
`AVAssetReaderOutput.h`

# Instance Methods

## initWithTrack:outputSettings:

Initializes an asset reader to wrap a specified track, with optional output settings.

```
- (id)initWithTrack:(AVAssetTrack *)track outputSettings:(NSDictionary
    *)outputSettings
```

**Parameters**

*track*

The track from which the reader should source sample buffers.

*outputSettings*

A dictionary of output settings to be used for sample output. Pass `nil` to receive samples in their original format as stored in the track.

You use keys from one of AVAudioSettings.h (linear PCM only), AVVideoSettings.h, or <CoreVideo/CVPixelBuffer.h>, depending on the media type and the output format you want.

**Return Value**

An asset reader wrapping *track*, using the setting defined by *outputSettings*.

**Discussion**

Initialization fails if the output settings cannot be used with the specified track.

`AVAssetReaderTrackOutput` does not currently support the AVAudioSettings.h keys `AVSampleRateKey` (page 395), `AVNumberOfChannelsKey` (page 395), or `AVChannelLayoutKey` (page 397).

For optimal performance when decompressing video, the requested pixel format should match what the decoder outputs natively to avoid unnecessary conversions. For H.264 use either `kCVPixelFormatType_420YpCbCr8BiPlanarVideoRange`, or `kCVPixelFormatType_420YpCbCr8BiPlanarFullRange` if the video is known to be full range. If the pixel buffers need to be in RGB for additional processing then `kCVPixelFormatType_32BGRA` is recommended.

**Availability**

Available in iOS 4.1 and later.

**Declared In**

`AVAssetReaderOutput.h`

# AVAssetReaderVideoCompositionOutput Class Reference

| | |
|---|---|
| **Inherits from** | AVAssetReaderOutput : NSObject |
| **Conforms to** | NSObject (NSObject) |
| **Framework** | /System/Library/Frameworks/AVFoundation.framework |
| **Availability** | Available in iOS 4.1 and later. |
| **Declared in** | AVAssetReaderOutput.h |
| **Companion guide** | AV Foundation Programming Guide |

## Overview

`AVAssetReaderVideoCompositionOutput` is a subclass of `AVAssetReaderOutput` you use to read video frames that have been composited together from the frames in one or more tracks of an `AVAssetReader` object's asset.

You can read the video frames composited from one or more asset tracks by adding an instance of `AVAssetReaderVideoCompositionOutput` to an `AVAssetReader` object using the `addOutput:` (page 57) method.

## Tasks

### Creating a Video Composition Output

`+ assetReaderVideoCompositionOutputWithVideoTracks:videoSettings:` (page 73)
Returns an instance of `AVAssetReaderVideoCompositionOutput` for reading composited video from the specified video tracks, using optional video settings.

`- initWithVideoTracks:videoSettings:` (page 74)
Initializes an instance of `AVAssetReaderVideoCompositionOutput` for reading composited video from the specified video tracks, using optional video settings.

## Properties



# Properties

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

## videoComposition

The video composition to use for the output.

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

**Discussion**
The value is an `AVVideoComposition` object that can be used to specify the visual arrangement of video frames read from each source track over the timeline of the source asset.

See `AVVideoComposition` for options for configuring a video composition.

**Availability**
Available in iOS 4.1 and later.

**Declared In**
`AVAssetReaderOutput.h`

## videoSettings

The video settings used by the output. (read-only)

```
@property(nonatomic, readonly) NSDictionary *videoSettings
```

**Discussion**
A value of `nil` indicates that the receiver will return video frames in a convenient uncompressed format, with properties determined according to the properties of the receiver's video tracks.

The dictionary’s keys are from `<CoreVideo/CVPixelBuffer.h>`.

**Availability**
Available in iOS 4.1 and later.

**Declared In**
`AVAssetReaderOutput.h`

### videoTracks

The tracks from which the output reads composited video. (read-only)

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

**Discussion**
The array contains `AVAssetTrack` objects owned by the target asset reader's asset.

**Availability**
Available in iOS 4.1 and later.

**Declared In**
`AVAssetReaderOutput.h`

## Class Methods

### assetReaderVideoCompositionOutputWithVideoTracks:videoSettings:

Returns an instance of `AVAssetReaderVideoCompositionOutput` for reading composited video from the specified video tracks, using optional video settings.

```
+ (AVAssetReaderVideoCompositionOutput
    *)assetReaderVideoCompositionOutputWithVideoTracks:(NSArray *)videoTracks
    videoSettings:(NSDictionary *)videoSettings
```

**Parameters**

*`videoTracks`*

An array of `AVAssetTrack` objects from which to read video frames for compositing.

Each track must be one of the tracks owned by the target asset reader's asset and must be of media type `AVMediaTypeVideo`.

*`videoSettings`*

A dictionary of video settings to be used for sample output, or `nil` if you want to receive decoded samples in a convenient uncompressed format, with properties determined according to the properties of the specified video tracks.

You use keys from `<CoreVideo/CVPixelBuffer.h>`, depending on the output format you want.

**Return Value**
An instance of `AVAssetReaderVideoCompositionOutput` wrapping *`videoTracks`*, using the settings specified by *`videoSettings`*, or `nil` if initialization failed.

**Discussion**
Initialization will fail if the video settings cannot be used with the specified video tracks.

**Availability**
Available in iOS 4.1 and later.

**Declared In**
`AVAssetReaderOutput.h`

# Instance Methods

## initWithVideoTracks:videoSettings:

Initializes an instance of `AVAssetReaderVideoCompositionOutput` for reading composited video from the specified video tracks, using optional video settings.

```
- (id)initWithVideoTracks:(NSArray *)videoTracks videoSettings:(NSDictionary
    *)videoSettings
```

**Parameters**

*`videoTracks`*

An array of `AVAssetTrack` objects from which to read video frames for compositing.

Each track must be one of the tracks owned by the target asset reader's asset and must be of media type `AVMediaTypeVideo`.

*`videoSettings`*

A dictionary of video settings to be used for sample output, or `nil` if you want to receive decoded samples in a convenient uncompressed format, with properties determined according to the properties of the specified video tracks.

You use keys from `<CoreVideo/CVPixelBuffer.h>`, depending on the output format you want.

**Return Value**
An instance of `AVAssetReaderVideoCompositionOutput` wrapping *`videoTracks`*, using the settings specified by *`videoSettings`*, or `nil` if initialization failed.

**Discussion**
Initialization will fail if the video settings cannot be used with the specified video tracks.

**Availability**
Available in iOS 4.1 and later.

**Declared In**
`AVAssetReaderOutput.h`

# AVAssetTrack Class Reference

| | |
|---|---|
| **Inherits from** | NSObject |
| **Conforms to** | NSCopying<br>AVAsynchronousKeyValueLoading<br>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 78) *property*
The asset of which the track is a part. (read-only)

`trackID` (page 83) *property*
The persistent unique identifier for this track of the asset. (read-only)

`mediaType` (page 80) *property*
The media type for the track. (read-only)

- `hasMediaCharacteristic:` (page 83)
Returns a Boolean value that indicates whether the track references media with the specified media characteristic.

`formatDescriptions` (page 79) *property*
The formats of media samples referenced by the track. (read-only)

`enabled` (page 78) *property*
Indicates whether the track is enabled according to state stored in its container or construct. (read-only)

`selfContained` (page 82) *property*
Indicates whether the track references sample data only within its storage container. (read-only)

`totalSampleDataLength` (page 83) *property*
The total number of bytes of sample data required by the track. (read-only)

## Temporal Properties

`timeRange` (page 83) *property*
The time range of the track within the overall timeline of the asset. (read-only)

`naturalTimeScale` (page 81) *property*
A timescale in which time values for the track can be operated upon without extraneous numerical conversion. (read-only)

`estimatedDataRate` (page 79) *property*
The estimated data rate of the media data referenced by the track, in bits per second. (read-only)

## Track Language Properties

`languageCode` (page 80) *property*
The language associated with the track, as an ISO 639-2/T language code. (read-only)

`extendedLanguageTag` (page 79) *property*
The language tag associated with the track, as an RFC 4646 language tag. (read-only)

## Visual Characteristics

`naturalSize` (page 80) *property*
The natural dimensions of the media data referenced by the track. (read-only)

`preferredTransform` (page 81) *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 82) *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 81) *property*
The frame rate of the track, in frames per second. (read-only)

### Track Segments

`segments` (page 82) *property*
The time mappings from the track's media samples to the timeline of the track. (read-only)

- `segmentForTrackTime:` (page 85)
The track segment that corresponds to the specified track time.

- `samplePresentationTimeForTrackTime:` (page 84)
Maps the specified track time through the appropriate time mapping and returns the resulting sample presentation time.

### Managing Metadata

`commonMetadata` (page 78) *property*
An array of `AVMetadataItem` objects for each common metadata key for which a value is available. (read-only)

- `metadataForFormat:` (page 84)
An array of metadata items, one for each metadata item in the container of the specified format.

`availableMetadataFormats` (page 78) *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 80)

**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 80)
`- hasMediaCharacteristic:` (page 83)

**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 79)

**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 83)
`@property formatDescriptions` (page 79)

**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 85)

**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 80)

`@property formatDescriptions` (page 79)

**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 78) 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 82)

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

`empty` (page 87) *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`

# AVAssetWriter Class Reference

| | |
|---|---|
| **Inherits from** | NSObject |
| **Conforms to** | NSObject (NSObject) |
| **Framework** | /System/Library/Frameworks/AVFoundation.framework |
| **Availability** | Available in iOS 4.1 and later. |
| **Declared in** | AVAssetWriter.h |

## Overview

You use an `AVAssetWriter` object to write media data to a new file of a specified audiovisual container type, such as a QuickTime movie file or an MPEG-4 file, with support for automatic interleaving of media data for multiple concurrent tracks.

You can get the media data for one or more assets from from instances of `AVAssetReader` or even from outside the AV Foundation API set. Media data is presented to `AVAssetWriter` for writing in the form of CMSampleBuffers (see *CMSampleBuffer Reference*). Sequences of sample data appended to the asset writer inputs are considered to fall within "sample-writing sessions." You must call `startSessionAtSourceTime:` (page 99) to begin one of these sessions.

Using `AVAssetWriter`, you can optionally re-encode media samples as they are written. You can also optionally write metadata collections to the output file.

You can only use a given instance of `AVAssetWriter` once to write to a single file. If you want to write to files multiple times, you must use a new instance of `AVAssetWriter` each time.

## Tasks

### Creating an Asset Writer

`+ assetWriterWithURL:fileType:error:` (page 95)
Returns an asset writer for writing to the file identified by a given URL in a format specified by a given UTI.

- `initWithURL:fileType:error:` (page 98)
  Initializes an asset writer for writing to the file identified by a given URL in a format specified by a given UTI.

`availableMediaTypes` (page 91) *property*
The media types for which inputs can be added (read-only)

## Writing Data

- `startWriting` (page 100)
  Tells the writer to start writing its output.
- `finishWriting` (page 98)
  Completes the writing of the output file.
- `cancelWriting` (page 97)
  Instructs the writer to cancel writing.

`outputURL` (page 94) *property*
The URL to which output is directed. (read-only)

`outputFileType` (page 93) *property*
The file format of the writer's output. (read-only)

`error` (page 91) *property*
If the receiver's status is `AVAssetWriterStatusFailed`, describes the error that caused the failure. (read-only)

`status` (page 95) *property*
The status of writing samples to the receiver's output file. (read-only)

## Managing Inputs

`inputs` (page 92) *property*
The asset writer inputs associated with the asset writer. (read-only)

- `addInput:` (page 96)
  Adds an input to the receiver.
- `canAddInput:` (page 96)
  Returns a Boolean value that indicates whether a given input can be added to the receiver.

## Managing Session Time

- `startSessionAtSourceTime:` (page 99)
  Initiates a sample-writing session for the output asset.

- `endSessionAtSourceTime:` (page 97)
  Concludes an explicit sample-writing session.

---

### Configuring Output

- `canApplyOutputSettings:forMediaType:` (page 97)
  Returns a Boolean value that indicates whether give output settings are supported for a specified media type.

`metadata` (page 92) *property*
The collection of metadata for association with the asset and for carriage in the output file.

`movieFragmentInterval` (page 92) *property*
The time to elapse between writing movie fragments.

`movieTimeScale` (page 93) *property*
Specifies the asset-level time scale to be used.

`shouldOptimizeForNetworkUse` (page 94) *property*
Indicates whether the output file should be written in way that makes it more suitable for playback over a network.

## Properties

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

### availableMediaTypes

The media types for which inputs can be added (read-only)

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

**Discussion**
Some media types may not be accepted within the type of file with which the writer was initialized.

**Availability**
Available in iOS 4.1 and later.

**Declared In**
`AVAssetWriter.h`

### error

If the receiver's status is `AVAssetWriterStatusFailed`, describes the error that caused the failure. (read-only)

```
@property(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 write to its output file. If the receiver's `status` (page 95) is not `AVAssetWriterStatusFailed`, the value of this property is `nil`.

**Availability**
Available in iOS 4.1 and later.

**Declared In**
`AVAssetWriter.h`

## inputs

The asset writer inputs associated with the asset writer. (read-only)

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

**Discussion**
The array contains `AVAssetWriterInput` objects.

**Availability**
Available in iOS 4.1 and later.

**See Also**
`- addInput:` (page 96)

**Declared In**
`AVAssetWriter.h`

## metadata

The collection of metadata for association with the asset and for carriage in the output file.

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

**Discussion**
The array contains `AVMetadataItem` objects.

**Special Considerations**
You cannot set the value after writing has started.

**Availability**
Available in iOS 4.1 and later.

**Declared In**
`AVAssetWriter.h`

## movieFragmentInterval

The time to elapse between writing movie fragments.

```
@property(nonatomic) CMTime movieFragmentInterval
```

**Discussion**
This property only applies to the QuickTime movie file type.

Sometimes a write operation may be unexpectedly interrupted (because a process crashes, for example). By using movie fragments, such a partially-written QuickTime movie file can be successfully opened and played up to the largest multiple of *movieFragmentInterval* smaller than the point at which the write operation was interrupted.

The default value is `kCMTimeInvalid`, which means that movie fragments should not be used, that only a movie atom describing all of the media in the file should be written.

**Special Considerations**

You cannot set the value after writing has started.

**Availability**
Available in iOS 4.1 and later.

**Declared In**
`AVAssetWriter.h`

## movieTimeScale

Specifies the asset-level time scale to be used.

```
@property(nonatomic) CMTimeScale movieTimeScale
```

**Discussion**
For file types that contain a `moov` atom, such as QuickTime Movie files, specifies the asset-level time scale to be used.

The default value is `0`, which indicates that you should choose a convenient value, if applicable.

**Special Considerations**

You cannot set the value after writing has started.

**Availability**
Available in iOS 4.3 and later.

**Declared In**
`AVAssetWriter.h`

## outputFileType

The file format of the writer's output. (read-only)

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

**Discussion**
The format is identified by the UTI, specified when the writer is initialized.

**Availability**
Available in iOS 4.1 and later.

**See Also**
`+ assetWriterWithURL:fileType:error:` (page 95)
`- initWithURL:fileType:error:` (page 98)

**Declared In**
`AVAssetWriter.h`

## outputURL

The URL to which output is directed. (read-only)

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

**Discussion**
The URL is the same as that specified when the writer is initialized.

**Availability**
Available in iOS 4.1 and later.

**See Also**
`+ assetWriterWithURL:fileType:error:` (page 95)
`- initWithURL:fileType:error:` (page 98)

**Declared In**
`AVAssetWriter.h`

## shouldOptimizeForNetworkUse

Indicates whether the output file should be written in way that makes it more suitable for playback over a network.

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

**Discussion**
When the value of this property is `YES`, the output file will be written in such a way that playback can start after only a small amount of the file is downloaded.

The default value is `NO`.

**Special Considerations**
You cannot set the value after writing has started.

**Availability**
Available in iOS 4.1 and later.

**Declared In**
`AVAssetWriter.h`

### status

The status of writing samples to the receiver's output file. (read-only)

```
@property(readonly) AVAssetWriterStatus status
```

**Discussion**
The value of this property is an `AVAssetWriterStatus` constant that indicates whether writing is in progress, has completed successfully, has been canceled, or has failed. If an attempt to append samples fails, you can check the value of this property to determine why no more samples could be written.

**Availability**
Available in iOS 4.1 and later.

**Declared In**
`AVAssetWriter.h`

## Class Methods

### assetWriterWithURL:fileType:error:

Returns an asset writer for writing to the file identified by a given URL in a format specified by a given UTI.

```
+ (AVAssetWriter *)assetWriterWithURL:(NSURL *)outputURL fileType:(NSString
    *)outputFileType error:(NSError **)outError
```

**Parameters**

*outputURL*
The location of the file to be written. The URL must be a file URL.

*outputFileType*
The UTI-identified format of the file to be written.

For example, `AVFileTypeQuickTimeMovie` for a QuickTime movie file, `AVFileTypeMPEG4` for an MPEG-4 file, and `AVFileTypeAMR` for an adaptive multi-rate audio format file.

*outError*
If initialization of the asset writer fails, upon return contains an error object that describes the problem.

**Return Value**
An asset writer for writing to the file identified by *URL* in the format specified by *outputFileType*, or `nil` if the writer could not be initialized.

**Discussion**
Writing will fail if a file already exists at *URL*. UTIs for container formats that can be written are declared in `AVMediaFormat.h`.

**Availability**
Available in iOS 4.1 and later.

**See Also**
- `initWithURL:fileType:error:` (page 98)
- `@property outputURL` (page 94)
- `@property outputFileType` (page 93)

**Declared In**
AVAssetWriter.h

# Instance Methods

## addInput:

Adds an input to the receiver.

```
- (void)addInput:(AVAssetWriterInput *)input
```

**Parameters**

*input*

The asset writer input to be added.

**Discussion**
Inputs are created with a media type and output settings. These both must be compatible with the receiver.

**Special Considerations**

You cannot add inputs after writing has started.

**Availability**
Available in iOS 4.1 and later.

**See Also**
- canAddInput: (page 96)

**Declared In**
AVAssetWriter.h

## canAddInput:

Returns a Boolean value that indicates whether a given input can be added to the receiver.

```
- (BOOL)canAddInput:(AVAssetWriterInput *)input
```

**Parameters**

*input*

The asset writer input to be tested.

**Return Value**
YES if input can be added, otherwise NO.

**Discussion**
You cannot add an input that accepts media data of a type that is not compatible with the receiver, or with output settings that are not compatible with the receiver.

**Availability**
Available in iOS 4.1 and later.

**Declared In**
AVAssetWriter.h

## canApplyOutputSettings:forMediaType:

Returns a Boolean value that indicates whether give output settings are supported for a specified media type.

```
- (BOOL)canApplyOutputSettings:(NSDictionary *)outputSettings forMediaType:(NSString *)mediaType
```

**Parameters**

*outputSettings*

The output settings to validate.

*mediaType*

The media type for which the output settings are validated.

**Return Value**

`YES` if the output settings in *outputSettings* are supported for *mediaType*, otherwise `NO`.

**Discussion**

You can use this method to test, for example, whether video output settings that specify H.264 compression will fail (as would be the case if the container format for which the writer was initialized does not support the carriage of H.264-compressed video).

**Availability**

Available in iOS 4.1 and later.

**Declared In**

`AVAssetWriter.h`

## cancelWriting

Instructs the writer to cancel writing.

```
- (void)cancelWriting
```

**Discussion**

This method blocks until writing is canceled.

**Availability**

Available in iOS 4.1 and later.

**Declared In**

`AVAssetWriter.h`

## endSessionAtSourceTime:

Concludes an explicit sample-writing session.

```
- (void)endSessionAtSourceTime:(CMTime)endTime
```

**Parameters**

*endTime*

The ending asset time for the sample-writing session, in the timeline of the source samples.

**Discussion**
You may invoke this method to complete a session you began by invoking `startSessionAtSourceTime:` (page 99).

You do not *need* to call this method; if you call `finishWriting` (page 98) without calling this method, the session's effective end time will be the latest end timestamp of the session's samples (that is, no samples will be edited out at the end).

The *endTime* defines the moment on the timeline of source samples at which the session ends. In the case of the QuickTime movie file format, each sample-writing session's startTime…endTime pair corresponds to a period of movie time into which the session's samples are inserted. Samples with later timestamps will be still be added to the media but will be edited out of the movie. So if the first session has duration `D1 = endTime - startTime`, it will be inserted into the movie at movie time 0 through D1; the second session would be inserted into the movie at movie time D1 through D1+D2, and so on.

It is legal to have a session with no samples; this will cause creation of an empty edit of the prescribed duration.

**Availability**
Available in iOS 4.1 and later.

**See Also**
`- startSessionAtSourceTime:` (page 99)

**Declared In**
`AVAssetWriter.h`

## finishWriting

Completes the writing of the output file.

```
- (BOOL)finishWriting
```

**Return Value**
`YES` if writing can be finished, otherwise `NO`.

**Discussion**
This method blocks until writing is finished. When this method returns successfully, the file being written by the receiver is complete and ready to use. You can check the values of the `status` (page 95) and `error` (page 91) properties for more information on why writing could not be finished.

**Availability**
Available in iOS 4.1 and later.

**Declared In**
`AVAssetWriter.h`

## initWithURL:fileType:error:

Initializes an asset writer for writing to the file identified by a given URL in a format specified by a given UTI.

```
- (id)initWithURL:(NSURL *)outputURL fileType:(NSString *)outputFileType
    error:(NSError **)outError
```

**Parameters**

*outputURL*

The location of the file to be written. The URL must be a file URL.

*outputFileType*

The UTI-identified format of the file to be written.

For example, `AVFileTypeQuickTimeMovie` for a QuickTime movie file, `AVFileTypeMPEG4` for an MPEG-4 file, and `AVFileTypeAMR` for an adaptive multi-rate audio format file.

*outError*

If initialization of the asset writer fails, upon return contains an error object that describes the problem.

**Return Value**

An asset writer for writing to the file identified by *URL* in the format specified by *outputFileType*, or `nil` if the writer could not be initialized.

**Discussion**

Writing will fail if a file already exists at *URL*. UTIs for container formats that can be written are declared in `AVMediaFormat.h`.

**Availability**

Available in iOS 4.1 and later.

**See Also**

`+ assetWriterWithURL:fileType:error:` (page 95)

`@property outputURL` (page 94)

`@property outputFileType` (page 93)

**Declared In**

`AVAssetWriter.h`

## startSessionAtSourceTime:

Initiates a sample-writing session for the output asset.

```
- (void)startSessionAtSourceTime:(CMTime)startTime
```

**Parameters**

*startTime*

The starting asset time for the sample-writing session, in the timeline of the source samples.

**Discussion**

Sequences of sample data appended to the asset writer inputs are considered to fall within "sample-writing sessions." *You must call this method to begin one of these sessions.*

Each writing session has a start time which, where allowed by the file format being written, defines the mapping from the timeline of source samples onto the file's timeline. In the case of the QuickTime movie file format, the first session begins at movie time 0, so a sample appended with timestamp T will be played at movie time (T-*startTime*). Samples with timestamps before *startTime* will still be added to the output media but will be edited out of the movie. If the earliest buffer for an input is later than *startTime*, an empty edit will be inserted to preserve synchronization between tracks of the output asset.

**Special Considerations**

It is an error to invoke this method twice in a row without invoking `endSessionAtSourceTime:` (page 97): in between.

**Availability**
Available in iOS 4.1 and later.

**See Also**
`- endSessionAtSourceTime:` (page 97)

**Declared In**
`AVAssetWriter.h`

### startWriting

Tells the writer to start writing its output.

```
- (BOOL)startWriting
```

**Return Value**
`YES` if writing can be started, otherwise `NO`.

**Discussion**
You must call this method after all inputs have added and other configuration properties have been set to tell the receiver to prepare for writing. After invoking this method, you can start writing sessions using `startSessionAtSourceTime:` (page 99) and can write media samples using the methods provided by each of the writer's inputs.

`status` (page 95) signals the terminal state of the asset reader, and if a failure occurs, `error` (page ?) describes the failure.

**Availability**
Available in iOS 4.1 and later.

**See Also**
`@property status` (page 95)
`@property error` (page 91)
`- startSessionAtSourceTime:` (page 99)

**Declared In**
`AVAssetWriter.h`

## Constants

### AVAssetWriterStatus

Type for status constants. See "Status Constants" (page 101) for possible values.

```
typedef NSInteger AVAssetWriterStatus;
```

**Availability**
Available in iOS 4.1 and later.

**Declared In**
`AVAssetWriter.h`

## Status Constants

These constants are returned by the `status` (page 95) property to indicate whether it can successfully write samples to its output file.

```
enum {
    AVAssetWriterStatusUnknown = 0,
    AVAssetWriterStatusWriting,
    AVAssetWriterStatusCompleted,
    AVAssetWriterStatusFailed,
    AVAssetWriterStatusCancelled
};
```

### Constants

`AVAssetWriterStatusUnknown`

Available in iOS 4.1 and later.

Declared in `AVAssetWriter.h`.

`AVAssetWriterStatusWriting`

Available in iOS 4.1 and later.

Declared in `AVAssetWriter.h`.

`AVAssetWriterStatusCompleted`

Available in iOS 4.1 and later.

Declared in `AVAssetWriter.h`.

`AVAssetWriterStatusFailed`

Available in iOS 4.1 and later.

Declared in `AVAssetWriter.h`.

`AVAssetWriterStatusCancelled`

Available in iOS 4.1 and later.

Declared in `AVAssetWriter.h`.

# AVAssetWriterInput Class Reference

| | |
|---|---|
| **Inherits from** | NSObject |
| **Conforms to** | NSObject (NSObject) |
| **Framework** | /System/Library/Frameworks/AVFoundation.framework |
| **Availability** | Available in iOS 4.1 and later. |
| **Declared in** | AVAssetWriterInput.h |
| **Companion guide** | AV Foundation Programming Guide |

## Overview

You use an `AVAssetWriterInput` to append media samples packaged as CMSampleBuffer objects, or collections of metadata, to a single track of the output file of an `AVAssetWriter` object.

When there are multiple inputs, `AVAssetWriter` tries to write media data in an ideal interleaving pattern for efficiency in storage and playback. Each of its inputs signals its readiness to receive media data for writing according to that pattern via the value of `readyForMoreMediaData` (page 106). If `readyForMoreMediaData` is `YES`, an input can accept additional media data while maintaining appropriate interleaving. If media data is appended to an input after `readyForMoreMediaData` becomes `NO`, `AVAssetWriter` may need to write media data to its output without regard for ideal interleaving.

You can only append media data to an input while its `readyForMoreMediaData` property is `YES`.

- If you're writing media data from a non-real-time source, such as an instance of `AVAssetReader`, you should hold off on generating or obtaining more media data to append to an input when the value of `readyForMoreMediaData` is `NO`. To help with control of the supply of non-real-time media data, you can use `requestMediaDataWhenReadyOnQueue:usingBlock:` (page 110) to specify a block that the input should invoke whenever it's ready for input to be appended.
- If you're writing media data from a real-time source, you should set the input's `expectsMediaDataInRealTime` property to `YES` to ensure that the value of `readyForMoreMediaData` is calculated appropriately. When `expectsMediaDataInRealTime` is `YES`, `readyForMoreMediaData` will become `NO` only when the input cannot process media samples as quickly as they are being provided by the client. If `readyForMoreMediaData` becomes `NO` for a real-time source, the client may need to drop samples or consider reducing the data rate of appended samples.

The value of `readyForMoreMediaData` will often change from `NO` to `YES` asynchronously, as previously-supplied media data is processed and written to the output. It is possible for all of an asset writer's inputs temporarily to return `NO` for `readyForMoreMediaData`.

# Tasks

## Creating an Asset Writer

+ `assetWriterInputWithMediaType:outputSettings:` (page 107)
  Returns a new input of the specified media type to receive sample buffers for writing to the output file.

- `initWithMediaType:outputSettings:` (page 109)
  Initialized a new input of the specified media type to receive sample buffers for writing to the output file.

## Adding Samples

- `appendSampleBuffer:` (page 108)
  Appends samples to the receiver.

`expectsMediaDataInRealTime` (page 105) *property*
  Indicates whether the input should tailor its processing of media data for real-time sources.

`readyForMoreMediaData` (page 106) *property*
  Indicates the readiness of the input to accept more media data. (read-only)

- `markAsFinished` (page 109)
  Tells the writer that no more buffers will be appended to this input.

- `requestMediaDataWhenReadyOnQueue:usingBlock:` (page 110)
  Instructs the receiver to invoke a block repeatedly, at its convenience, in order to gather media data for writing to the output.

## Inspecting a Writer

`mediaType` (page 105) *property*
  The media type of the samples that can be appended to the input. (read-only)

`metadata` (page 106) *property*
  The collection of track-level metadata for association with the asset and for carriage in the output file.

`transform` (page 107) *property*
  The transform specified in the output file as the preferred transformation of the visual media data for display purposes.

`outputSettings` (page 106) *property*
  The settings used for encoding the media appended to the output. (read-only)

`mediaTimeScale` (page 105) *property*
Specifies the media time scale to be used

# Properties

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

## expectsMediaDataInRealTime

Indicates whether the input should tailor its processing of media data for real-time sources.

```
@property(nonatomic) BOOL expectsMediaDataInRealTime
```

**Discussion**
If you are appending media data to an input from a real-time source, such as an `AVCaptureOutput`, you should set `expectsMediaDataInRealTime` to `YES`. This will ensure that `readyForMoreMediaData` (page 106) is calculated appropriately for real-time usage.

**Availability**
Available in iOS 4.1 and later.

**Declared In**
`AVAssetWriterInput.h`

## mediaTimeScale

Specifies the media time scale to be used

```
@property(nonatomic) CMTimeScale mediaTimeScale
```

**Discussion**
For file types that support media time scales, such as QuickTime Movie files, specifies the media time scale to be used.

The default value is `0`, which indicates that you should choose a convenient value, if applicable.

You cannot set this property after writing has started.

**Availability**
Available in iOS 4.3 and later.

**Declared In**
`AVAssetWriterInput.h`

## mediaType

The media type of the samples that can be appended to the input. (read-only)

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

**Discussion**
The value of this property is one of the media type strings defined in `AVMediaFormat.h`.

**Availability**
Available in iOS 4.1 and later.

**Declared In**
`AVAssetWriterInput.h`

## metadata

The collection of track-level metadata for association with the asset and for carriage in the output file.

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

**Discussion**
The array contains `AVMetadataItem` objects representing the collection of track-level metadata to be written in the output file.

You cannot set this property after writing on the receiver's asset writer has started.

**Availability**
Available in iOS 4.1 and later.

**Declared In**
`AVAssetWriterInput.h`

## outputSettings

The settings used for encoding the media appended to the output. (read-only)

```
@property(nonatomic, readonly) NSDictionary *outputSettings
```

**Discussion**
A value of `nil` specifies that appended samples should not be re-encoded.

**Availability**
Available in iOS 4.1 and later.

**See Also**
`- initWithMediaType:outputSettings:` (page 109)
`+ assetWriterInputWithMediaType:outputSettings:` (page 107)

**Declared In**
`AVAssetWriterInput.h`

## readyForMoreMediaData

Indicates the readiness of the input to accept more media data. (read-only)

```
@property(nonatomic, readonly, getter=isReadyForMoreMediaData) BOOL
    readyForMoreMediaData
```

**Discussion**
This property is observable using key-value observing (see *Key-Value Observing Programming Guide*). Observers should not assume that they will be notified of changes on a specific thread.

**Availability**
Available in iOS 4.1 and later.

**Declared In**
`AVAssetWriterInput.h`

### transform

The transform specified in the output file as the preferred transformation of the visual media data for display purposes.

```
@property(nonatomic) CGAffineTransform transform
```

**Discussion**
If no value is specified, the identity transform is used.

**Availability**
Available in iOS 4.1 and later.

**Declared In**
`AVAssetWriterInput.h`

## Class Methods

### assetWriterInputWithMediaType:outputSettings:

Returns a new input of the specified media type to receive sample buffers for writing to the output file.

```
+ (AVAssetWriterInput *)assetWriterInputWithMediaType:(NSString *)mediaType
    outputSettings:(NSDictionary *)outputSettings
```

**Parameters**

*mediaType*

The media type of samples that will be accepted by the input.

Media types are defined in `AVMediaFormat.h`.

*outputSettings*

The settings used for encoding the media appended to the output. Pass `nil` to specify that appended samples should not be re-encoded.

Audio output settings keys are defined in `AVAudioSettings.h`. Video output settings keys are defined in `AVVideoSettings.h`. Video output settings with keys from `<CoreVideo/CVPixelBuffer.h>` are not currently supported.

**Return Value**
A new input of the specified media type to receive sample buffers for writing to the output file.

**Discussion**
Each new input accepts data for a new track of the asset writer's output file. You add an input to an asset writer using the `AVAssetWriter` method `addInput:` (page 96).

Passing `nil` for *`outputSettings`* instructs the input to pass through appended samples, doing no processing before they are written to the output file. This is useful if, for example, you are appending buffers that are already in a desirable compressed format. However, passthrough is currently supported only when writing to QuickTime Movie files (i.e. the `AVAssetWriter` was initialized with `AVFileTypeQuickTimeMovie`). For other file types, you must specify non-`nil` output settings.

**Availability**
Available in iOS 4.1 and later.

**Declared In**
`AVAssetWriterInput.h`

# Instance Methods

## appendSampleBuffer:

Appends samples to the receiver.

```
- (BOOL)appendSampleBuffer:(CMSampleBufferRef)sampleBuffer
```

**Parameters**

*`sampleBuffer`*
The CMSampleBuffer to be appended.

**Return Value**
`YES` if sampleBuffer as appended successfully, otherwise `NO`.

**Discussion**
The timing information in the sample buffer, considered relative to the time passed to the asset writer's `startSessionAtSourceTime:` (page 99) will be used to determine the timing of those samples in the output file.

Do not modify *`sampleBuffer`* or its contents after you have passed it to this method.

**Availability**
Available in iOS 4.1 and later.

**See Also**
`- requestMediaDataWhenReadyOnQueue:usingBlock:` (page 110)

**Declared In**
`AVAssetWriterInput.h`

## initWithMediaType:outputSettings:

Initialized a new input of the specified media type to receive sample buffers for writing to the output file.

```
- (id)initWithMediaType:(NSString *)mediaType outputSettings:(NSDictionary
    *)outputSettings
```

**Parameters**

*mediaType*

The media type of samples that will be accepted by the input.

Media types are defined in `AVMediaFormat.h`.

*outputSettings*

The settings used for encoding the media appended to the output. Pass `nil` to specify that appended samples should not be re-encoded.

Audio output settings keys are defined in `AVAudioSettings.h`. Video output settings keys are defined in `AVVideoSettings.h`. Video output settings with keys from `<CoreVideo/CVPixelBuffer.h>` are not currently supported.

**Return Value**
An input of the specified media type initialized to receive sample buffers for writing to the output file.

**Discussion**
Each new input accepts data for a new track of the asset writer's output file. You add an input to an asset writer using the `AVAssetWriter` method `addInput:` (page 96).

Passing `nil` for *outputSettings* instructs the input to pass through appended samples, doing no processing before they are written to the output file. This is useful if, for example, you are appending buffers that are already in a desirable compressed format. However, passthrough is currently supported only when writing to QuickTime Movie files (i.e. the `AVAssetWriter` was initialized with `AVFileTypeQuickTimeMovie`). For other file types, you must specify non-`nil` output settings.

**Availability**
Available in iOS 4.1 and later.

**Declared In**
`AVAssetWriterInput.h`

## markAsFinished

Tells the writer that no more buffers will be appended to this input.

```
- (void)markAsFinished
```

**Discussion**
If you are monitoring each input's `expectsMediaDataInRealTime` (page 105) value to keep the output file well interleaved, it is important to call this method when you have finished adding buffers to a track. This is necessary to prevent other inputs from stalling, as they may otherwise wait forever for that input's media data, attempting to complete the ideal interleaving pattern.

**Availability**
Available in iOS 4.1 and later.

**Declared In**
`AVAssetWriterInput.h`

## requestMediaDataWhenReadyOnQueue:usingBlock:

Instructs the receiver to invoke a block repeatedly, at its convenience, in order to gather media data for writing to the output.

```
- (void)requestMediaDataWhenReadyOnQueue:(dispatch_queue_t)queue usingBlock:(void
    (^)(void))block
```

### Parameters

*queue*

The queue on which *block* should be invoked.

*block*

The block the input should invoke to obtain media data.

### Discussion

The block should append media data to the input either until the input's `readyForMoreMediaData` (page 106) property becomes `NO` or until there is no more media data to supply (at which point it may choose to mark the input as finished using `markAsFinished` (page 109)). The block should then exit. After the block exits, if the input has not been marked as finished, once the input has processed the media data it has received and becomes ready for more media data again, it will invoke the block again in order to obtain more.

A typical use of this method, with a block that supplies media data to an input while respecting the input's `readyForMoreMediaData` property, might look like this:

```
[myAVAssetWriterInput requestMediaDataWhenReadyOnQueue:myInputSerialQueue
usingBlock:^{
        while ([myAVAssetWriterInput isReadyForMoreMediaData])
        {
            CMSampleBufferRef nextSampleBuffer = [self
copyNextSampleBufferToWrite];
            if (nextSampleBuffer)
            {
                [myAVAssetWriterInput appendSampleBuffer:nextSampleBuffer];
                CFRelease(nextSampleBuffer);
            }
            else
            {
                [myAVAssetWriterInput markAsFinished];
                break;
            }
        }
    }];
```

You should not use this method with a push-style buffer source, such as `AVCaptureAudioDataOutput` or `AVCaptureVideoDataOutput`, because such a combination will typically require intermediate queueing of buffers. Instead, this method is better suited to a pull-style buffer source such as an `AVAssetReaderOutput` object.

When using a push-style buffer source, it is generally better to immediately append each buffer to the asset writer input, directly as it is received using `appendSampleBuffer:` (page 108). Using this strategy, it is often possible to avoid having to queue up buffers in between the buffer source and the asset writer input. Note that many of these push-style buffer sources also produce buffers in real-time, in which case you should set `expectsMediaDataInRealTime` (page 105) to `YES`.

### Availability

Available in iOS 4.1 and later.

### See Also

- `markAsFinished` (page 109)

### Declared In

`AVAssetWriterInput.h`

# AVAssetWriterInputPixelBufferAdaptor Class Reference

| | |
|---|---|
| **Inherits from** | NSObject |
| **Conforms to** | NSObject (NSObject) |
| **Framework** | /System/Library/Frameworks/AVFoundation.framework |
| **Availability** | Available in iOS 4.1 and later. |
| **Declared in** | AVAssetWriterInput.h |
| **Companion guide** | AV Foundation Programming Guide |

## Overview

You use an `AVAssetWriterInputPixelBufferAdaptor` to append video samples packaged as CVPixelBuffer objects to a single `AVAssetWriterInput` object.

Instances of `AVAssetWriterInputPixelBufferAdaptor` provide a CVPixelBufferPool that you can use to allocate pixel buffers for writing to the output file. Using the provided pixel buffer pool for buffer allocation is typically more efficient than appending pixel buffers allocated using a separate pool.

## Tasks

### Creating an Adaptor

`+ assetWriterInputPixelBufferAdaptorWithAssetWriterInput:sourcePixelBufferAttributes:` (page 115)

Returns a new pixel buffer adaptor to receive pixel buffers for writing to the output file.

`- initWithAssetWriterInput:sourcePixelBufferAttributes:` (page 117)

Initializes a new pixel buffer adaptor to receive pixel buffers for writing to the output file.

### Adding a Pixel Buffer

`- appendPixelBuffer:withPresentationTime:` (page 116)
Appends a pixel buffer to the receiver.

### Inspecting a Pixel Buffer Adaptor

`assetWriterInput` (page 114) *property*
The asset writer input to which the adaptor should append pixel buffers. (read-only)

`pixelBufferPool` (page 114) *property*
A pixel buffer pool that will vend and efficiently recycle CVPixelBuffer objects that can be appended to the receiver. (read-only)

`sourcePixelBufferAttributes` (page 115) *property*
The pixel buffer attributes of pixel buffers that will be vended by the adaptor's CVPixelBufferPool. (read-only)

## Properties

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

### assetWriterInput

The asset writer input to which the adaptor should append pixel buffers. (read-only)

```
@property(nonatomic, readonly) AVAssetWriterInput *assetWriterInput
```

**Availability**
Available in iOS 4.1 and later.

**Declared In**
`AVAssetWriterInput.h`

### pixelBufferPool

A pixel buffer pool that will vend and efficiently recycle CVPixelBuffer objects that can be appended to the receiver. (read-only)

```
@property(nonatomic, readonly) CVPixelBufferPoolRef pixelBufferPool
```

**Discussion**
For maximum efficiency, you should create CVPixelBuffer objects for `appendPixelBuffer:withPresentationTime:` (page 116) by using this pool with the `CVPixelBufferPoolCreatePixelBuffer` function.

This property is `NULL` before the first call to `startSessionAtSourceTime:` (page 99) on the associated `AVAssetWriter` object.

This property is key value observable.

**Availability**
Available in iOS 4.1 and later.

**See Also**
`- appendPixelBuffer:withPresentationTime:` (page 116)

**Declared In**
`AVAssetWriterInput.h`

### sourcePixelBufferAttributes

The pixel buffer attributes of pixel buffers that will be vended by the adaptor's CVPixelBufferPool. (read-only)

```
@property(nonatomic, readonly) NSDictionary *sourcePixelBufferAttributes
```

**Discussion**
The value of this property is a dictionary containing pixel buffer attributes keys defined in `<CoreVideo/CVPixelBuffer.h>`.

**Availability**
Available in iOS 4.1 and later.

**Declared In**
`AVAssetWriterInput.h`

## Class Methods

### assetWriterInputPixelBufferAdaptorWithAssetWriterInput: sourcePixelBufferAttributes:

Returns a new pixel buffer adaptor to receive pixel buffers for writing to the output file.

```
+ (AVAssetWriterInputPixelBufferAdaptor
    *)assetWriterInputPixelBufferAdaptorWithAssetWriterInput:(AVAssetWriterInput
    *)input sourcePixelBufferAttributes:(NSDictionary *)sourcePixelBufferAttributes
```

**Parameters**

*input*

The asset writer input to which the receiver should append pixel buffers.

Currently, only asset writer inputs that accept media data of type `AVMediaTypeVideo` (page 400) can be used to initialize a pixel buffer adaptor.

It is an error to pass a sample buffer input that is already attached to another instance of `AVAssetWriterInputPixelBufferAdaptor`.

*sourcePixelBufferAttributes*

The attributes of pixel buffers that will be vended by the input's CVPixelBufferPool.

Pixel buffer attributes keys for the pixel buffer pool are defined in `<CoreVideo/CVPixelBuffer.h>`. To take advantage of the improved efficiency of appending buffers created from the adaptor's pixel buffer pool, you should specify pixel buffer attributes that most closely accommodate the source format of the video frames being appended.

Pass `nil` if you do not need a pixel buffer pool for allocating buffers.

**Return Value**

A new pixel buffer adaptor to receive pixel buffers for writing to the output file.

**Discussion**

To specify the pixel format type, the *pixelBufferAttributes* dictionary should contain a value for `kCVPixelBufferPixelFormatTypeKey`. For example, use `[NSNumber numberWithInt:kCVPixelFormatType_32BGRA]` for 8-bit-per-channel BGRA, or use `[NSNumber numberWithInt:kCVPixelFormatType_420YpCbCr8BiPlanarVideoRange]` for 2-plane YCbCr.

**Availability**

Available in iOS 4.1 and later.

**Declared In**

`AVAssetWriterInput.h`

# Instance Methods

## appendPixelBuffer:withPresentationTime:

Appends a pixel buffer to the receiver.

```
- (BOOL)appendPixelBuffer:(CVPixelBufferRef)pixelBuffer
    withPresentationTime:(CMTime)presentationTime
```

**Parameters**

*pixelBuffer*

The CVPixelBuffer to be appended.

*presentationTime*

The presentation time for the pixel buffer to be appended. This time is considered relative to the time passed to `startSessionAtSourceTime:` (page 99) to determine the timing of the frame in the output file.

**Return Value**

`YES` if the pixel buffer was successfully appended, otherwise `NO`.

### Discussion
If the operation was unsuccessful, you might invoke the `AVAssetWriter` object's `finishWriting` (page 98) method in order to save a partially completed asset.

### Special Considerations
Do not modify a CVPixelBuffer or its contents after you have passed it to this method.

### Availability
Available in iOS 4.1 and later.

### See Also
`@property pixelBufferPool` (page 114)

### Declared In
`AVAssetWriterInput.h`

## initWithAssetWriterInput:sourcePixelBufferAttributes:
Initializes a new pixel buffer adaptor to receive pixel buffers for writing to the output file.

```
- (id)initWithAssetWriterInput:(AVAssetWriterInput *)input
    sourcePixelBufferAttributes:(NSDictionary *)sourcePixelBufferAttributes
```

### Parameters
*input*

The asset writer input to which the receiver should append pixel buffers.

Currently, only asset writer inputs that accept media data of type `AVMediaTypeVideo` (page 400) can be used to initialize a pixel buffer adaptor.

It is an error to pass a sample buffer input that is already attached to another instance of `AVAssetWriterInputPixelBufferAdaptor`.

*sourcePixelBufferAttributes*

The attributes of pixel buffers that will be vended by the input's CVPixelBufferPool.

Pixel buffer attributes keys for the pixel buffer pool are defined in `<CoreVideo/CVPixelBuffer.h>`. To take advantage of the improved efficiency of appending buffers created from the adaptor's pixel buffer pool, you should specify pixel buffer attributes that most closely accommodate the source format of the video frames being appended.

Pass `nil` if you do not need a pixel buffer pool for allocating buffers.

### Return Value
A pixel buffer adaptor initialized to receive pixel buffers for writing to the output file.

### Discussion
To specify the pixel format type, the *pixelBufferAttributes* dictionary should contain a value for `kCVPixelBufferPixelFormatTypeKey`. For example, use `[NSNumber numberWithInt:kCVPixelFormatType_32BGRA]` for 8-bit-per-channel BGRA, or use `[NSNumber numberWithInt:kCVPixelFormatType_420YpCbCr8BiPlanarVideoRange]` for 2-plane YCbCr.

### Availability
Available in iOS 4.1 and later.

### Declared In
`AVAssetWriterInput.h`

# AVAudioMix Class Reference

| | |
|---|---|
| **Inherits from** | NSObject |
| **Conforms to** | NSCopying<br>NSMutableCopying<br>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 119) *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<br>NSMutableCopying<br>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 122) 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 122) *property*
The trackID of the audio track to which the parameters should be applied. (read-only)

### Getting Volume Ramps

`- getVolumeRampForTime:startVolume:endVolume:timeRange:` (page 122)
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<br>AQOfflineRenderTest<br>iPhoneExtAudioFileConvertTest<br>Metronome<br>oalTouch |

## 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 126).

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 131)) 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 133)
  Initializes and returns an audio player for playing a designated sound file.
- `initWithData:error:` (page 134)
  Initializes and returns an audio player for playing a designated memory buffer.

## Configuring and Controlling Playback

- `play` (page 135)
  Plays a sound asynchronously.
- `playAtTime:` (page 136)
  Plays a sound asynchronously, starting at a specified point in the audio output device's timeline.
- `pause` (page 134)
  Pauses playback; sound remains ready to resume playback from where it left off.
- `stop` (page 138)
  Stops playback and undoes the setup needed for playback.
- `prepareToPlay` (page 137)
  Prepares the audio player for playback by preloading its buffers.

`playing` (page 131) *property*
A Boolean value that indicates whether the audio player is playing (`YES`) or not (`NO`). (read-only)

`volume` (page 132) *property*
The playback gain for the audio player, ranging from 0.0 through 1.0.

`pan` (page 131) *property*
The audio player's stereo pan position.

`numberOfLoops` (page 130) *property*
The number of times a sound will return to the beginning, upon reaching the end, to repeat playback.

`delegate` (page 128) *property*
The delegate object for the audio player.

`settings` (page 131) *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 130) *property*
The number of audio channels in the sound associated with the audio player. (read-only)

`duration` (page 129) *property*
Returns the total duration, in seconds, of the sound associated with the audio player. (read-only)

`currentTime` (page 128) *property*
The playback point, in seconds, within the timeline of the sound associated with the audio player.

`deviceCurrentTime` (page 129) *property*
The time value, in seconds, of the audio output device. (read-only)

`url` (page 132) *property*
The URL for the sound associated with the audio player. (read-only)

`data` (page 128) *property*
The data object containing the sound associated with the audio player. (read-only)

## Using Audio Level Metering

`meteringEnabled` (page 130) *property*
A Boolean value that indicates the audio-level metering on/off state for the audio player.

- `averagePowerForChannel:` (page 133)
Returns the average power for a given channel, in decibels, for the sound being played.

- `peakPowerForChannel:` (page 135)
Returns the peak power for a given channel, in decibels, for the sound being played.

- `updateMeters` (page 138)
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 135) 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 129)
`@property duration` (page 129)

**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 132)

**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 136) 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 128)
`- playAtTime:` (page 136)

**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 128)

**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 133)
- `peakPowerForChannel:` (page 135)
- `updateMeters` (page 138)

**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 138) 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 379) 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 397))
- Encoder bit rate (`AVEncoderBitRateKey` (page 397))
- Audio data format (`AVFormatIDKey` (page 395))
- Channel count (`AVNumberOfChannelsKey` (page 395))
- Sample rate (`AVSampleRateKey` (page 395))

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 128)

**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 138) method before calling this method.

**Availability**

Available in iOS 2.2 and later.

**See Also**

`@property meteringEnabled` (page 130)

`- peakPowerForChannel:` (page 135)

**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*

Pass in the address of a `nil`-initialized `NSError` object. If an error occurs, upon return the `NSError` object describes the error. If you do not want error information, pass in `NULL`.

**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 134)

**Related Sample Code**
AddMusic
AQOfflineRenderTest
iPhoneExtAudioFileConvertTest
Metronome
oalTouch

**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*

Pass in the address of a `nil`-initialized `NSError` object. If an error occurs, upon return the `NSError` object describes the error. If you do not want error information, pass in `NULL`.

**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 133)

**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 135)
- `prepareToPlay` (page 137)
- `stop` (page 138)

**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 138) method before calling this method.

**Availability**
Available in iOS 2.2 and later.

**See Also**
- `@property meteringEnabled` (page 130)
- `averagePowerForChannel:` (page 133)

**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 134)
- `playAtTime:` (page 136)
- `prepareToPlay` (page 137)
- `stop` (page 138)

**Related Sample Code**
AddMusic
AQOfflineRenderTest
iPhoneExtAudioFileConvertTest
Metronome
oalTouch

**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 129) 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 129) 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 134)
- `play` (page 135)
- `prepareToPlay` (page 137)
- `stop` (page 138)

**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 134)
- `play` (page 135)
- `stop` (page 138)

**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 128) 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 134)
- `play` (page 135)
- `prepareToPlay` (page 137)

**Related Sample Code**
oalTouch

**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 133) or `peakPowerForChannel:` (page 135).

**Availability**
Available in iOS 2.2 and later.

**See Also**
`@property meteringEnabled` (page 130)

**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 142) 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 143)
Initializes and returns an audio recorder.

## Configuring and Controlling Recording

- `prepareToRecord` (page 145)
  Creates an audio file and prepares the system for recording.
- `record` (page 145)
  Starts or resumes recording.
- `recordForDuration:` (page 146)
  Records for a specified duration of time.
- `pause` (page 144)
  Pauses a recording.
- `stop` (page 146)
  Stops recording and closes the audio file.

`delegate` (page 141) *property*
The delegate object for the audio recorder.

- `deleteRecording` (page 143)
  Deletes a recorded audio file.

## Managing Information About a Recording

`recording` (page 142) *property*
A Boolean value that indicates whether the audio recorder is recording (`YES`), or not (`NO`).

`url` (page 142) *property*
The URL for the audio file associated with the audio recorder.

`currentTime` (page 141) *property*
The time, in seconds, since the beginning of the recording.

`settings` (page 142) *property*
The audio settings for the audio recorder.

## Using Audio Level Metering

`meteringEnabled` (page 141) *property*
A Boolean value that indicates whether audio-level metering is enabled (`YES`), or not (`NO`).

- `updateMeters` (page 146)
  Refreshes the average and peak power values for all channels of an audio recorder.
- `peakPowerForChannel:` (page 144)
  Returns the peak power for a given channel, in decibels, for the sound being recorded.
- `averagePowerForChannel:` (page 142)
  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 145) 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 146) method before calling this method.

**Availability**

Available in iOS 3.0 and later.

**See Also**

`@property meteringEnabled` (page 141)

`- peakPowerForChannel:` (page 144)

**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*

Pass in the address of a `nil`-initialized `NSError` object. If an error occurs, upon return the `NSError` object describes the error. If you do not want error information, pass in `NULL`.

**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 145) 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 146) method immediately before calling this method.

**Availability**
Available in iOS 3.0 and later.

**See Also**
`- averagePowerForChannel:` (page 142)
`@property meteringEnabled` (page 141)

**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 143) 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 145). 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 145), 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 145), 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 142) or peakPowerForChannel: (page 144).

**Availability**
Available in iOS 3.0 and later.

**See Also**
`@property meteringEnabled` (page 141)

**Declared In**
`AVAudioRecorder.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 |
| **Companion guide** | AV Foundation Programming Guide |

## 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.

An instance of `AVCaptureAudioDataOutput` produces audio sample buffers suitable for processing using other media APIs. It passes the sample buffers to its delegate using the `captureOutput:didOutputSampleBuffer:fromConnection:` (page 385) method. To get the sample buffers, you implement `captureOutput:didOutputSampleBuffer:fromConnection:` in the delegate object.

## Tasks

### Managing the Delegate

- `setSampleBufferDelegate:queue:` (page 150)
  Sets the delegate that will accept captured buffers and dispatch queue on which the delegate will be called.

`sampleBufferDelegate` (page 150) *property*
The capture object's delegate.

`sampleBufferCallbackQueue` (page 150) *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 delegate that will accept captured buffers and dispatch queue on which the delegate will be called.

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

**Parameters**

*`sampleBufferDelegate`*

An object conforming to the `AVCaptureAudioDataOutputSampleBufferDelegate` protocol that will receive sample buffers after they are captured..

*sampleBufferCallbackQueue*

You must pass a serial dispatch to guarantee that audio samples will be delivered in order.

The value may not be `NULL`, except when setting the *sampleBufferDelegate* to `nil`.

**Discussion**
When a new audio sample buffer is captured it is vended to the sample buffer delegate using the `captureOutput:didOutputSampleBuffer:fromConnection:` (page 385) delegate method. All delegate methods are called on the specified dispatch queue.

If the queue is blocked when new samples are captured, those samples will be automatically dropped when they become sufficiently late. This allows you to process existing samples 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 samples.

If you need to minimize the chances of samples being dropped, you should specify a queue on which a sufficiently small amount of processing is being done outside of receiving sample buffers. However, you migrate extra processing to another queue, you are responsible for ensuring that memory usage does not grow without bound from samples 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.

**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 155) *property*
Indicates whether the connection is enabled.

`active` (page 154) *property*
Indicates whether the connection is active. (read-only)

`inputPorts` (page 155) *property*
The connection's input ports. (read-only)

`output` (page 155) *property*
The connection's output port. (read-only)

`audioChannels` (page 154) *property*
An array of `AVCaptureAudioChannel` objects. (read-only)

`videoMirrored` (page 156) *property*
Indicates whether the video is mirrored.

`supportsVideoMirroring` (page 155) *property*
Indicates whether the connection supports mirroring of the video. (read-only)

`videoOrientation` (page 156) *property*
Indicates the orientation of the video.

`supportsVideoOrientation` (page 156) *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 173). 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 169)
Returns an array containing the available capture devices on the system.

`+ deviceWithUniqueID:` (page 170)
Returns the device with a given ID.

`+ defaultDeviceWithMediaType:` (page 169)
Returns the default device used to capture data of a given media type.

`+ devicesWithMediaType:` (page 169)
Returns an array containing the devices able to capture data of a given media type

## Focus Settings

`focusMode` (page 165) *property*
The device's focus mode.

- `isFocusModeSupported:` (page 172)
Returns a Boolean value that indicates whether the given focus mode is supported.

`focusPointOfInterest` (page 165) *property*
The point of interest for focusing.

`focusPointOfInterestSupported` (page 166) *property*
Indicates whether the device supports a point of interest for focus. (read-only)

`adjustingFocus` (page 162) *property*
Indicates whether the device is currently adjusting its focus setting. (read-only)

## Exposure Settings

`adjustingExposure` (page 162) *property*
Indicates whether the device is currently adjusting its exposure setting. (read-only)

`exposureMode` (page 163) *property*
The exposure mode for the device.

- `isExposureModeSupported:` (page 171)
Returns a Boolean value that indicates whether the given exposure mode is supported.

`exposurePointOfInterest` (page 164) *property*
The point of interest for exposure.

`exposurePointOfInterestSupported` (page 164) *property*
Indicates whether the device supports a point of interest for exposure. (read-only)

## Flash Settings

`hasFlash` (page 166) *property*
Indicates whether the capture device has a flash. (read-only)

`flashMode` (page 164) *property*
The current flash mode.

- `isFlashModeSupported:` (page 171)
Returns a Boolean value that indicates whether the given flash mode is supported.

## White Balance Settings

- `isWhiteBalanceModeSupported:` (page 173)
  Returns a Boolean value that indicates whether the given white balance mode is supported.

`whiteBalanceMode` (page 168) *property*
The current white balance mode.

`adjustingWhiteBalance` (page 163) *property*
Indicates whether the devise is currently adjusting the white balance. (read-only)

## Torch Mode Settings

`hasTorch` (page 166) *property*
A Boolean value that specifies whether the capture device has a torch. (read-only)

- `isTorchModeSupported:` (page 172)
  Returns a Boolean value that indicates whether the given torch mode is supported.

`torchMode` (page 168) *property*
The current torch mode.

## Device Characteristics

`connected` (page 163) *property*
Indicates whether the device is currently connected. (read-only)

`position` (page 167) *property*
(read-only)

- `hasMediaType:` (page 170)

`modelID` (page 167) *property*
(read-only)

`localizedName` (page 167) *property*
(read-only)

`uniqueID` (page 168) *property*
(read-only)

- `supportsAVCaptureSessionPreset:` (page 174)

### Locking the Device

- `lockForConfiguration:` (page 173)
  Attempts to acquire a lock on the capture device.
- `unlockForConfiguration` (page 174)
  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 163)
- `@property exposurePointOfInterest` (page 164)

**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 166)
- `@property focusPointOfInterest` (page 165)
- `isFocusModeSupported:` (page 172)

**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 173)
- `@property whiteBalanceMode` (page 168)

**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 178) for possible values.

**Availability**
Available in iOS 4.0 and later.

**See Also**
- `isExposureModeSupported:` (page 171)
- `@property adjustingExposure` (page 162)
- `@property exposurePointOfInterest` (page 164)

- lockForConfiguration: (page 173)

**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 162)
@property exposurePointOfInterestSupported (page 164)

**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 164)
- isExposureModeSupported: (page 171)
@property exposureMode (page 163)

**Declared In**
AVCaptureDevice.h

## flashMode

The current flash mode.

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

**Discussion**
See "Flash Modes" (page 175) for possible values.

**Availability**
Available in iOS 4.0 and later.

**See Also**
`@property hasFlash` (page 166)
`- isFlashModeSupported:` (page 171)
`- lockForConfiguration:` (page 173)

**Declared In**
`AVCaptureDevice.h`

## focusMode

The device's focus mode.

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

**Discussion**
See "Focus Modes" (page 177) for possible values.

**Availability**
Available in iOS 4.0 and later.

**See Also**
`@property focusPointOfInterestSupported` (page 166)
`@property focusPointOfInterest` (page 165)
`- isFocusModeSupported:` (page 172)
`- lockForConfiguration:` (page 173)

**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 166)

**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 165)
`- isFocusModeSupported:` (page 172)

**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 164)
`- isFlashModeSupported:` (page 171)

**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 168)
`- isTorchModeSupported:` (page 172)

**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 167)
`@property uniqueID` (page 168)

**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 167)
`@property uniqueID` (page 168)

**Declared In**
`AVCaptureDevice.h`

## position

(read-only)

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

**Discussion**
See "Capture Device Position" (page 175) 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 176) for possible values.

**Availability**
Available in iOS 4.0 and later.

**See Also**
@property hasTorch (page 166)
- isTorchModeSupported: (page 172)
- lockForConfiguration: (page 173)

**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 167)

**Declared In**
AVCaptureDevice.h

## whiteBalanceMode

The current white balance mode.

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

**Discussion**
See "White Balance Modes" (page 179) for possible values.

**Availability**
Available in iOS 4.0 and later.

**See Also**
- isWhiteBalanceModeSupported: (page 173)
@property adjustingWhiteBalance (page 163)
- lockForConfiguration: (page 173)

**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 168)

**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 178) 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 163)
@property exposurePointOfInterestSupported (page 164)
**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 175) 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 166)
`@property flashMode` (page 164)

**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 177) 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 165)
`@property adjustingFocus` (page 162)

**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 176) 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 168)

**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 179) 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 168)

**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 165), exposureMode (page 163), 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 174)

**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 173)

**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 175) 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 175) 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 176) 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 177) 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 178) 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 179) 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 184)
  Starts recording to a given URL.
- `stopRecording` (page 184)
  Stops recording.

`recording` (page 184) *property*
Indicates whether recording is in progress.

### Configuration

`maxRecordedDuration` (page 182) *property*
The longest duration allowed for the recording.

`maxRecordedFileSize` (page 182) *property*
The maximum file size allowed for the recording.

`minFreeDiskSpaceLimit` (page 183) *property*
The minimum available free disk space that must be available for recording to continue.

---

### Information About Output

`outputFileURL` (page 183) *property*
The URL to which output is directed. (read-only)

`recordedDuration` (page 183) *property*
The total duration recorded to the current output file. (read-only)

`recordedFileSize` (page 183) *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 183) 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 183) 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 183) 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 197) 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 187) *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 258) to ensure that an array of track segments conforms to this rule.

## Tasks

### Movie Configuration

`movieFragmentInterval` (page 190) *property*
Indicates the number of seconds of output that are written per fragment.

`metadata` (page 190) *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 197).

## Tasks

### Accessing Connections

`connections` (page 192) *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 200) to start the flow of data from the inputs to the outputs, and `stopRunning` (page 201) to stop the flow. You use the `sessionPreset` (page 196) property to customize the quality of the output.

# Tasks

## Managing Inputs and Outputs

`inputs` (page 195) *property*
The capture session's inputs. (read-only)

`outputs` (page 196) *property*
The capture session's outputs. (read-only)

- `addInput:` (page 197)
Adds a given input to the session.

- `addOutput:` (page 197)
Adds a given output to the session.

- `canAddInput:` (page 198)
Returns a Boolean value that indicates whether a given input can be added to the session.

- `canAddOutput:` (page 198)
Returns a Boolean value that indicates whether a given output can be added to the session.

- `removeInput:` (page 200)
Removes a given input.

- `removeOutput:` (page 200)
Removes a given output.

## Managing Running State

- `startRunning` (page 200)
Tells the receiver to start running.

- `stopRunning` (page 201)
Tells the receiver to stop running.

`running` (page 196) *property*
Indicates whether the receiver is running. (read-only)

`interrupted` (page 195) *property*
Indicates whether the receiver has been interrupted. (read-only)

## Configuration Change

- `beginConfiguration` (page 197)
Indicates the start of a set of configuration changes to be made atomically.

- `commitConfiguration` (page 199)
  Commits a set of configuration changes.

---

### Managing Session Presets

`sessionPreset` (page 196) *property*
The capture session's preset.

- `canSetSessionPreset:` (page 199)
  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 197)
- `canAddInput:` (page 198)
- `removeInput:` (page 200)

**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 197)
- `canAddOutput:` (page 198)
- `removeOutput:` (page 200)

**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 202).

**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 198)
- `addOutput:` (page 197)
- `removeInput:` (page 200)

**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 198)
- `addInput:` (page 197)
- `removeOutput:` (page 200)

**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 199) 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 196), or configure individual capture input or output properties. No changes are actually made until you invoke `commitConfiguration` (page 199), at which time they are applied together.

**Availability**
Available in iOS 4.0 and later.

**See Also**
- `commitConfiguration` (page 199)

**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 197)

**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 197)

**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 202).

**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 197).

**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 197)

**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 197)

**Declared In**
AVCaptureSession.h

## startRunning

Tells the receiver to start running.

```
- (void)startRunning
```

**Discussion**
startRunning and stopRunning (page 201) are asynchronous operations. If an error occurs occur during a capture session, you receive an AVCaptureSessionRuntimeErrorNotification (page 203).

**Availability**
Available in iOS 4.0 and later.

**See Also**
- `stopRunning` (page 201)

**Declared In**
`AVCaptureSession.h`

### stopRunning

Tells the receiver to stop running.

```
- (void)stopRunning
```

**Discussion**
`startRunning` (page 200) and `stopRunning` are asynchronous operations. If an error occurs occur during a capture session, you receive an `AVCaptureSessionRuntimeErrorNotification` (page 203).

**Availability**
Available in iOS 4.0 and later.

**See Also**
- `startRunning` (page 200)

**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 203).

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 202).

**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 208)
Initiates a still image capture and returns immediately.

### Image Configuration

`outputSettings` (page 206) *property*
The compression settings for the output.

`availableImageDataCVPixelFormatTypes` (page 206) *property*
The supported image pixel formats that can be specified in `outputSettings` (page 206). (read-only)

`availableImageDataCodecTypes` (page 206) *property*
The supported image codec formats that can be specified in `outputSettings` (page 206). (read-only)

### Image Format Conversion

`+ jpegStillImageNSDataRepresentation:` (page 207)
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 206). (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 402) in the `outputSettings` (page 206) 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 206). (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 206) 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 402) 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 |
| **Companion guide** | AV Foundation Programming Guide |

## Overview

`AVCaptureVideoDataOutput` is a concrete sub-class of `AVCaptureOutput` you use, via its delegate, 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. It passes the frames to its delegate using the `captureOutput:didOutputSampleBuffer:fromConnection:` method. To get the frames, you implement `captureOutput:didOutputSampleBuffer:fromConnection:` in the delegate object.

## Tasks

### Configuration

`videoSettings` (page 211) *property*
The compression settings for the output.

`minFrameDuration` (page 210) *property*
The minimum frame duration.

`alwaysDiscardsLateVideoFrames` (page 210) *property*
Indicates whether video frames are dropped if they arrive late.

### Managing the Delegate

- `setSampleBufferDelegate:queue:` (page 212)
  Sets the sample buffer delegate and the queue on which callbacks should be invoked.

`sampleBufferDelegate` (page 211) *property*
The capture object's delegate.

`sampleBufferCallbackQueue` (page 211) *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 212).

**Availability**
Available in iOS 4.0 and later.

**See Also**
`- setSampleBufferDelegate:queue:` (page 212)
`@property sampleBufferDelegate` (page 211)

**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 212).

**Availability**
Available in iOS 4.0 and later.

**See Also**
`- setSampleBufferDelegate:queue:` (page 212)
`@property sampleBufferCallbackQueue` (page 211)

**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`. Supported pixel formats are `kCVPixelFormatType_420YpCbCr8BiPlanarVideoRange`, `kCVPixelFormatType_420YpCbCr8BiPlanarFullRange` and `kCVPixelFormatType_32BGRA`, except on iPhone 3G, where the supported pixel formats are `kCVPixelFormatType_422YpCbCr8` and `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`*

An object conforming to the `AVCaptureVideoDataOutputSampleBufferDelegate` protocol that will receive sample buffers after they are captured.

*`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.

The sampleBufferCallbackQueue parameter may not be `NULL`, except when setting the *`sampleBufferDelegate`* to `nil`.

**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 invoked 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 210) 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 210) 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 211)

`@property sampleBufferCallbackQueue` (page 211)

**Declared In**
`AVCaptureOutput.h`

# AVCaptureVideoPreviewLayer Class Reference

| | |
|---|---|
| **Inherits from** | CALayer : NSObject |
| **Conforms to** | NSCoding (CALayer)<br>CAMediaTiming (CALayer)<br>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 219)
Initializes a preview layer with a given capture session.

`+ layerWithSession:` (page 218)
Returns a preview layer initialized with a given capture session.

### Layer Configuration

`orientation` (page 217) *property*
The layer's orientation.

`orientationSupported` (page 217) *property*
Indicates whether the layer display supports changing the orientation. (read-only)

`mirrored` (page 216) *property*
Indicates whether the layer display is mirrored.

`mirroringSupported` (page 217) *property*
Indicates whether the layer display supports mirroring. (read-only)

`automaticallyAdjustsMirroring` (page 216) *property*
Indicates whether the layer display automatically adjusts mirroring.

`videoGravity` (page 218) *property*
Indicates how the video is displayed within a player layer's bounds rect.

`session` (page 218) *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 216) 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<br>NSCopying (AVAsset)<br>AVAsynchronousKeyValueLoading (AVAsset)<br>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 222) *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)<br>AVAsynchronousKeyValueLoading (AVAssetTrack)<br>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 224) *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 the low-level representation of a composition to storage formats of your choosing and to reconstitute them from storage.

## Tasks

### Creating a Segment

`+ compositionTrackSegmentWithTimeRange:` (page 227)
Returns a composition track segment that presents an empty track segment.

`- initWithTimeRange:` (page 228)
Initializes a track segment that presents an empty track segment.

`+ compositionTrackSegmentWithURL:trackID:sourceTimeRange:targetTimeRange:` (page 227)
Returns a composition track segment that presents a portion of a file referenced by a given URL.

`- initWithURL:trackID:sourceTimeRange:targetTimeRange:` (page 229)
Initializes a track segment that presents a portion of a file referenced by a given URL.

### Segment Properties

`sourceURL` (page 227) *property*
The container file of the media presented by the track segment. (read-only)

`sourceTrackID` (page 226) *property*
The track ID of the container file of the media presented by the track segment. (read-only)

`empty` (page 226) *property*
Indicates whether the segment is empty. (read-only)

## Properties

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

### empty

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

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

**Discussion**
An empty segment has a valid target time range but `sourceURL` (page 227) is `nil` and the source start time is `kCMTimeInvalid`; all other fields are undefined.

**Availability**
Available in iOS 4.1 and later.

**Declared In**
`AVCompositionTrackSegment.h`

### 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 229) 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 229) 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<br>NSMutableCopying<br>AVAsynchronousKeyValueLoading<br>NSObject (NSObject) |
| **Framework** | /System/Library/Frameworks/AVFoundation.framework |
| **Availability** | Available in iOS 4.0 and later. |
| **Declared in** | AVMetadataItem.h |
| **Companion guide** | AV Foundation Programming Guide |

## Overview

An `AVMetadataItem` object represents an item of metadata associated with an audiovisual asset or with one of its tracks. To create metadata items for your own assets, you use the mutable subclass, `AVMutableMetadataItem`.

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.

You can load values of a metadata item "lazily" using the methods from the `AVAsynchronousKeyValueLoading` protocol (see "Asynchronous Loading" (page 232)). `AVAsset` and other classes in turn provide their metadata lazily so 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 237) and `metadataItemsFromArray:withKey:keySpace:` (page 236) respectively.

# Tasks

## Filtering Metadata Arrays

`+ metadataItemsFromArray:withKey:keySpace:` (page 236)
Returns from a given array an array of metadata items that match a specified key or key space.

`+ metadataItemsFromArray:withLocale:` (page 237)
Returns from a given array an array of metadata items that match a specified locale.

## Keys and Key Spaces

`key` (page 234) *property*
The metadata item's key. (read-only)

`keySpace` (page 235) *property*
The key space of metadata item's key. (read-only)

`commonKey` (page 233) *property*
The common key of the metadata item. (read-only)

## Asynchronous Loading

`- loadValuesAsynchronouslyForKeys:completionHandler:` (page 238)
Tells the receiver to load the values of any of the specified keys that are not already loaded.

`- statusOfValueForKey:error:` (page 238)
Reports whether the value for a given key is immediately available without blocking.

## Accessing Values

`value` (page 236) *property*
Provides the value of the metadata item. (read-only)

`time` (page 236) *property*
Indicates the timestamp of the metadata item. (read-only)

`duration` (page 234) *property*
The duration of the metadata item. (read-only)

`locale` (page 235) *property*
The locale of the metadata item. (read-only)

### Type Coercion



# 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`

## duration

The duration of the metadata item. (read-only)

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

**Availability**
Available in iOS 4.2 and later.

**Declared In**
`AVMetadataItem.h`

## extraAttributes

The additional attributes supplied by the metadata item. (read-only)

```
@property(readonly, copy) NSDictionary *extraAttributes
```

**Discussion**

**Availability**
Available in iOS 4.0 and later.

**Declared In**
`AVMetadataItem.h`

## key

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

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

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`

# Instance Methods

## loadValuesAsynchronouslyForKeys:completionHandler:

Tells the receiver to load the values of any of the specified keys that are not already loaded.

```
- (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**
For full discussion, see `AVAsynchronousKeyValueLoading`.

**Availability**
Available in iOS 4.3 and later.

**Declared In**
`AVMetadataItem.h`

## statusOfValueForKey:error:

Reports whether the value for a given key is immediately available without blocking.

```
- (AVKeyValueStatus)statusOfValueForKey:(NSString *)key error:(NSError **)outError
```

**Parameters**

*key*

The key whose status you want.

*outError*

If the status of the value for the key is `AVKeyValueStatusFailed`, upon return contains an `NSError` object that describes the failure that occurred.

**Return Value**
The current loading status of the value for *key*.

**Discussion**
For full discussion, see `AVAsynchronousKeyValueLoading`.

**Availability**
Available in iOS 4.3 and later.

**Declared In**
`AVMetadataItem.h`

# AVMutableAudioMix Class Reference

| | |
|---|---|
| **Inherits from** | AVAudioMix : NSObject |
| **Conforms to** | NSCopying (AVAudioMix)<br>NSMutableCopying (AVAudioMix)<br>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 240)
Returns a new mutable audio mix.

### Input Parameters

`inputParameters` (page 240) *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)<br>NSMutableCopying (AVAudioMixInputParameters)<br>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 242)
Returns a mutable input parameters object with no volume ramps and `trackID` (page 242) initialized to `kCMPersistentTrackID_Invalid`.

`+ audioMixInputParametersWithTrack:` (page 243)
Returns a mutable input parameters object for a given track.

### Managing the Track ID

`trackID` (page 242) *property*
The trackID of the audio track to which the parameters should be applied.

### Setting the Volume

- `setVolume:atTime:` (page 243)
  Sets the value of the audio volume at a specific time.
- `setVolumeRampFromStartVolume:toEndVolume:timeRange:` (page 243)
  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 242) initialized to `kCMPersistentTrackID_Invalid`.

```
+ (AVMutableAudioMixInputParameters *)audioMixInputParameters
```

**Return Value**
A mutable input parameters object with no volume ramps and `trackID` (page 242) 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 242) 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)<br>NSCopying (AVAsset)<br>AVAsynchronousKeyValueLoading (AVAsset)<br>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 248)
  Adds or extends an empty timeRange within all tracks of the composition.
- `insertTimeRange:ofAsset:atTime:error:` (page 249)
  Inserts all the tracks within a given time range of a specified asset into the receiver.

- `removeTimeRange:` (page 250)
  Removes a specified timeRange from all tracks of the composition.
- `scaleTimeRange:toDuration:` (page 251)
  Changes the duration of all tracks in a given time range.

---

### Creating a Mutable Composition

+ `composition` (page 247)
  Returns a new, empty, mutable composition.

---

### Managing Tracks

`tracks` (page 247) *property*
  An array of `AVMutableCompositionTrack` objects contained by the composition. (read-only)
- `addMutableTrackWithMediaType:preferredTrackID:` (page 248)
  Adds an empty track to the receiver.
- `removeTrack:` (page 251)
  Removes a specified track from the receiver.
- `mutableTrackCompatibleWithTrack:` (page 249)
  Returns a track in the receiver into which any time range of a given asset track can be inserted.

---

### Video Size

`naturalSize` (page 246) *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 249)

**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 249)

**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 251).

**Availability**
Available in iOS 4.0 and later.

**See Also**
- `insertEmptyTimeRange:` (page 248)

**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 248).

This method is similar to `compatibleTrackForCompositionTrack:` (page 353) (`AVAsset`).

**Availability**

Available in iOS 4.0 and later.

**See Also**

`- addMutableTrackWithMediaType:preferredTrackID:` (page 248)

**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 251)

**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 250)

**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)<br>AVAsynchronousKeyValueLoading (AVAssetTrack)<br>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 256)), you can test whether the segments meet the constraints using `validateTrackSegments:error:` (page 258).

## Tasks

### Managing Time Ranges

- `insertEmptyTimeRange:` (page 256)
  Adds or extends an empty time range within the receiver.
- `insertTimeRange:ofTrack:atTime:error:` (page 257)
  Inserts a time range of a source track.
- `removeTimeRange:` (page 257)
  Removes a specified time range from the receiver.
- `scaleTimeRange:toDuration:` (page 258)
  Changes the duration of a time range in the receiver.

`segments` (page 256) *property*
The composition track's array of track segments.

### Validating Segments

- `validateTrackSegments:error:` (page 258)
  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 255) *property*
The language associated with the track, as an ISO 639-2/T language code.

`extendedLanguageTag` (page 254) *property*
The language tag associated with the track, as an RFC 4646 language tag.

`naturalTimeScale` (page 255) *property*
The timescale in which time values for the track can be operated upon without extraneous numerical conversion.

`preferredTransform` (page 255) *property*
The preferred transformation of the visual media data for display purposes.

`preferredVolume` (page 256) *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 255)

**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 254)

**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 258) 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 258).

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 256) 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 256) 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)<br>NSMutableCopying (AVMetadataItem)<br>AVAsynchronousKeyValueLoading (AVMetadataItem)<br>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` you use to build collections of metadata to be written to asset files using `AVAssetExportSession`, `AVAssetWriter` or `AVAssetWriterInput`.

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 264)
Returns a new mutable metadata item.

### Key and Key Space

`key` (page 262) *property*
Indicates the metadata item's key.

`keySpace` (page 263) *property*
Indicates the key space of the metadata item's key.

### Values

`value` (page 264) *property*
Indicates the metadata item's value.

`time` (page 263) *property*
Indicates the metadata item's timestamp.

`duration` (page 262) *property*
Indicates the duration of the metadata item.

`locale` (page 263) *property*
Indicates the metadata item's locale.

`extraAttributes` (page 262) *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*.

### duration

Indicates the duration of the metadata item.

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

**Availability**
Available in iOS 4.2 and later.

**Declared In**
`AVMetadataItem.h`

### 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

# AVMutableTimedMetadataGroup Class Reference

| | |
|---|---|
| **Inherits from** | AVTimedMetadataGroup : NSObject |
| **Conforms to** | NSCopying (AVTimedMetadataGroup)<br>NSObject (NSObject) |
| **Framework** | /System/Library/Frameworks/AVFoundation.framework |
| **Availability** | Available in iOS 4.3 and later. |
| **Declared in** | AVTimedMetadataGroup.h |

## Overview

You use an `AVMutableTimedMetadataGroup` object to represent a mutable collection of metadata items.

## Tasks

### Modifying the Group

`timeRange` (page 266) *property*
The time range of the metadata.

`items` (page 265) *property*
The metadata items in the group.

## Properties

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

### items

The metadata items in the group.

```
@property(readwrite, copy) NSArray *items
```

**Discussion**
The array contains instances of `AVMetadataItem`.

**Availability**
Available in iOS 4.3 and later.

**Declared In**
`AVTimedMetadataGroup.h`

## timeRange

The time range of the metadata.

```
@property(readwrite) CMTimeRange timeRange
```

**Discussion**

**Availability**
Available in iOS 4.3 and later.

**Declared In**
`AVTimedMetadataGroup.h`

# AVMutableVideoComposition Class Reference

| | |
|---|---|
| **Inherits from** | AVVideoComposition : NSObject |
| **Conforms to** | NSCopying (AVVideoComposition)<br>NSMutableCopying (AVVideoComposition)<br>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 269)
Returns a new mutable video composition.

### Properties

`frameDuration` (page 268) *property*
The interval for which the video composition should render composed video frames.

`renderSize` (page 269) *property*
The size at which the video composition should render.

`renderScale` (page 269) *property*
The scale at which the video composition should render.

`instructions` (page 268) *property*
The video composition instructions.

`animationTool` (page 268) *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)<br>NSCopying (AVVideoCompositionInstruction)<br>NSMutableCopying (AVVideoCompositionInstruction)<br>NSObject (NSObject) |
| **Framework** | /System/Library/Frameworks/AVFoundation.framework |
| **Availability** | Available in iOS 4.0 and later. |
| **Declared in** | AVVideoComposition.h |
| **Companion guide** | AV Foundation Programming Guide |

## 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 273)
Returns a new mutable video composition instruction.

### Properties

`backgroundColor` (page 272) *property*
The background color of the composition.

`layerInstructions` (page 273) *property*
An array of instances of `AVVideoCompositionLayerInstruction` that specify how video frames from source tracks should be layered and composed.

`timeRange` (page 273) *property*
The time range during which the instruction is effective.

`enablePostProcessing` (page 272) *property*
Indicates whether post processing is required for the video composition 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 supported colors 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 `NULL`, the video compositor uses a default background color of opaque black.

**Availability**
Available in iOS 4.0 and later.

**Declared In**
`AVVideoComposition.h`

## enablePostProcessing

Indicates whether post processing is required for the video composition instruction.

```
@property(nonatomic, assign) BOOL enablePostProcessing
```

**Discussion**
If no post processing is required for the whole duration of the video composition instruction, set this property to `NO` to make the composition process more efficient.

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 the property value is `nil`, the output is a fill of the background color.

**Availability**
Available in iOS 4.0 and later.

**See Also**
`@property backgroundColor` (page 272)

**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)<br>NSCopying (AVVideoCompositionLayerInstruction)<br>NSMutableCopying (AVVideoCompositionLayerInstruction)<br>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 276)
Returns a new mutable video composition layer instruction.

`+ videoCompositionLayerInstructionWithAssetTrack:` (page 277)
Returns a new mutable video composition layer instruction for the given track.

### Track ID

`trackID` (page 276) *property*
The trackID of the source track to which the compositor will apply the instruction.

### Managing Properties

- `setOpacity:atTime:` (page 277)
  Sets a value of the opacity at a time within the time range of the instruction.
- `setOpacityRampFromStartOpacity:toEndOpacity:timeRange:` (page 278)
  Sets an opacity ramp to apply during a specified time range.
- `setTransform:atTime:` (page 278)
  Sets a value of the transform at a time within the time range of the instruction.
- `setTransformRampFromStartTransform:toEndTransform:timeRange:` (page 279)
  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 276) 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 276) 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.

## Tasks

### Creating a Player

`- initWithURL:` (page 289)
Initializes a new player to play a single audiovisual resource referenced by a given URL.

`+ playerWithURL:` (page 286)
Returns a new player to play a single audiovisual resource referenced by a given URL.

`- initWithPlayerItem:` (page 288)
Initializes a new player to play a given single audiovisual item.

`+ playerWithPlayerItem:` (page 285)
Returns a new player initialized to play a given single audiovisual item

## Managing Playback

- `play` (page 290)
  Begins playback of the current item.
- `pause` (page 289)
  Pauses playback.

`rate` (page 284) *property*
The current rate of playback.

`actionAtItemEnd` (page 283) *property*
The action to perform when an item has finished playing.

- `replaceCurrentItemWithPlayerItem:` (page 291)
  Replaces the player item with a new player item.

## Managing Time

- `currentTime` (page 288)
  Returns the current time of the current item.
- `seekToTime:` (page 291)
  Moves the playback cursor to a given time.
- `seekToTime:toleranceBefore:toleranceAfter:` (page 291)
  Moves the playback cursor within a specified time bound.

## Timed Observations

- `addPeriodicTimeObserverForInterval:queue:usingBlock:` (page 287)
  Requests invocation of a given block during playback to report changing time.
- `addBoundaryTimeObserverForTimes:queue:usingBlock:` (page 286)
  Requests invocation of a block when specified times are traversed during normal playback.
- `removeTimeObserver:` (page 290)
  Cancels a previously registered time observer.

## Configuring a Player

`closedCaptionDisplayEnabled` (page 283) *property*
Indicates whether the player uses closed captioning.

### Player Properties

`status` (page 285) *property*
Indicates whether the player can be used for playback. (read-only)

`error` (page 284) *property*
If the receiver's status is `AVPlayerStatusFailed` (page 293), this describes the error that caused the failure. (read-only)

`currentItem` (page 284) *property*
The player's current item. (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 293).

**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 293), 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 293), the value of this property is `nil`.

**Availability**
Available in iOS 4.0 and later.

**See Also**
`@property status` (page 285)

**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 290)
- `pause` (page 289)

**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 293), 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.

**See Also**
`@property error` (page 284)

**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 305) in `AVPlayerItem`).

**Availability**
Available in iOS 4.0 and later.

**See Also**
`- initWithPlayerItem:` (page 288)

**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 284).

**Availability**
Available in iOS 4.0 and later.

**See Also**
`- initWithURL:` (page 289)
`@property currentItem` (page 284)

**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 290).

**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 287)
- `removeTimeObserver:` (page 290)

`@property currentTime` (page 288)

**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 290).

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 290) will result in undefined behavior.

**Availability**
Available in iOS 4.0 and later.

**See Also**
- `addBoundaryTimeObserverForTimes:queue:usingBlock:` (page 286)
- `removeTimeObserver:` (page 290)

`@property currentTime` (page 288)

**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 287) or `addBoundaryTimeObserverForTimes:queue:usingBlock:` (page 286) instead.

**Availability**
Available in iOS 4.0 and later.

**See Also**
- `addPeriodicTimeObserverForInterval:queue:usingBlock:` (page 287)
- `addBoundaryTimeObserverForTimes:queue:usingBlock:` (page 286)

**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 305) in `AVPlayerItem`).

**Availability**

Available in iOS 4.0 and later.

**See Also**

`+ playerWithPlayerItem:` (page 285)

**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 284).

**Availability**

Available in iOS 4.0 and later.

**See Also**

`+ playerWithURL:` (page 286)

`@property currentItem` (page 284)

**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 284)

**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 284)

**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 287) or `addBoundaryTimeObserverForTimes:queue:usingBlock:` (page 286).

**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 287) and `addBoundaryTimeObserverForTimes:queue:usingBlock:` (page 286).

**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 284) 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 291).

**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 291) directly.

**Availability**

Available in iOS 4.0 and later.

**Declared In**

`AVPlayer.h`

# Constants

## AVPlayerStatus

Possible values of the `status` (page 285) 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 284) property.

Available in iOS 4.0 and later.

Declared in `AVPlayer.h`.

## AVPlayerActionAtItemEnd

You use these constants with `actionAtItemEnd` (page 283) to indicate the action a player should take when it finishes playing.

```
enum
{
    AVPlayerActionAtItemEndAdvance  = 0,
    AVPlayerActionAtItemEndPause    = 1,
    AVPlayerActionAtItemEndNone     = 2,
};
typedef NSInteger AVPlayerActionAtItemEnd;
```

### Constants

`AVPlayerActionAtItemEndAdvance`
Indicates that the player should advance to the next item, if there is one.

Available in iOS 4.1 and later.

Declared in `AVPlayer.h`.

`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<br>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 303) and `initWithURL:` (page 305)); 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 305)
  Prepares a player item with a given URL.

+ `playerItemWithURL:` (page 303)
  Returns a new player item, prepared to use a given URL.

- `initWithAsset:` (page 305)
  Initializes a new player item for a given asset.

+ `playerItemWithAsset:` (page 302)
  Returns a new player item for a given asset.

## Getting Information About an Item

`asset` (page 298) *property*
The underlying asset provided during initialization. (read-only)

`tracks` (page 302) *property*
An array of `AVPlayerItemTrack` objects. (read-only)

`status` (page 301) *property*
The status of the player item. (read-only)

`loadedTimeRanges` (page 299) *property*
The time ranges of the item that have been loaded. (read-only)

`presentationSize` (page 300) *property*
The size at which the visual portion of the item is presented by the player. (read-only)

`timedMetadata` (page 301) *property*
The timed metadata played most recently by the media stream. (read-only)

`seekableTimeRanges` (page 301) *property*
An array of time ranges within which it is possible to seek. (read-only)

`error` (page 298) *property*
If the receiver's status is `AVPlayerItemStatusFailed` (page 308), this describes the error that caused the failure. (read-only)

## Moving the Playhead

- `stepByCount:` (page 307)
  Moves the player's current item's current time forward or backward by a specified number of steps.

- `seekToTime:` (page 306)
  Moves the playback cursor to a given time.
- `seekToTime:toleranceBefore:toleranceAfter:` (page 307)
  Moves the playback cursor within a specified time bound.
- `seekToDate:` (page 306)
  Moves the playback cursor to a given date.

## Information About Playback

`playbackLikelyToKeepUp` (page 300) *property*
Indicates whether the item will likely play through without stalling (read-only)

`playbackBufferEmpty` (page 299) *property*
Indicates whether playback has consumed all buffered media and that playback will stall or end. (read-only)

`playbackBufferFull` (page 300) *property*
Indicates whether the internal media buffer is full and that further I/O is suspended. (read-only)

## Timing Information

- `currentTime` (page 304)
  Returns the current time of the item.
- `currentDate` (page 304)
  Returns the current time of the item as an `NSDate` object.

`forwardPlaybackEndTime` (page 299) *property*
The time at which forward playback ends.

`reversePlaybackEndTime` (page 300) *property*
The time at which reverse playback ends.

## Settings

`audioMix` (page 298) *property*
The audio mix parameters to be applied during playback.

`videoComposition` (page 302) *property*
The video composition settings to be applied during playback.

### Accessing Logs

- `accessLog` (page 303)
  Returns an object that represents a snapshot of the network access log.
- `errorLog` (page 304)
  Returns an object that represents a snapshot of the error log.

## 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 308), 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 308), 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

An array of time ranges within which it is possible to seek. (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 308).

**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 308).

If the URL contains valid data that can be used by the player item, the status later changes to `AVPlayerItemStatusReadyToPlay` (page 308).

If the URL contains no valid data or otherwise can't be used by the player item, the status later changes to `AVPlayerItemStatusFailed` (page 308).

**Availability**
Available in iOS 4.0 and later.

**See Also**
`@property status` (page 301)

**Declared In**
AVPlayerItem.h

## Instance Methods

### accessLog

Returns an object that represents a snapshot of the network access log.

```
- (AVPlayerItemAccessLog *)accessLog
```

**Return Value**
An object that represents a snapshot of the network access log. The returned value can be `nil`.

**Discussion**
If the method returns `nil`, there is no logging information currently available for the player item.

**Availability**
Available in iOS 4.3 and later.

**Declared In**
AVPlayerItem.h

## currentDate

Returns the current time of the item as an NSDate object.

```
- (NSDate *)currentDate
```

**Return Value**
The current time of the item as an NSDate object

**Discussion**

**Availability**
Available in iOS 4.3 and later.

**See Also**
- currentTime (page 304)

**Declared In**
AVPlayerItem.h

## 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.

**See Also**
- currentDate (page 304)

**Declared In**
AVPlayerItem.h

## errorLog

Returns an object that represents a snapshot of the error log.

```
- (AVPlayerItemErrorLog *)errorLog
```

**Return Value**
An object that represents a snapshot of the error log. The returned value can be nil.

**Discussion**
If the method returns `nil`, there is no logging information currently available for the player item.

**Availability**
Available in iOS 4.3 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 308).

If the URL contains valid data that can be used by the player item, the status later changes to `AVPlayerItemStatusReadyToPlay` (page 308).

If the URL contains no valid data or otherwise can't be used by the player item, the status later changes to `AVPlayerItemStatusFailed` (page 308).

**Availability**
Available in iOS 4.0 and later.

**See Also**
`@property status` (page 301)

**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 306)
- `seekToDate:` (page 306)

**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 307).

**Availability**
Available in iOS 4.0 and later.

**See Also**
- `seekToTime:toleranceBefore:toleranceAfter:` (page 307)

- seekToDate: (page 306)

**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 306) directly.

**Availability**
Available in iOS 4.0 and later.

**See Also**
- seekToTime: (page 306)
- seekToDate: (page 306)

**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 302)).

**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`.

## Notification Key

Key to retrieve information from a notification's user info dictionary.

```
extern NSString *const AVPlayerItemFailedToPlayToEndTimeErrorKey
```

**Constants**

`AVPlayerItemFailedToPlayToEndTimeErrorKey`
The key to retrieve an error object (`NSError`) from the user info dictionary of an `AVPlayerItemFailedToPlayToEndTimeNotification` (page 309) notification.
Available in iOS 4.3 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`

### AVPlayerItemFailedToPlayToEndTimeNotification

Posted when the item failed to play to its end time.

The notification's object is the item that finished playing.

The user info dictionary contains an error object that describes the problem—see `AVPlayerItemFailedToPlayToEndTimeErrorKey` (page 308).

> **Important:** This notification may be posted on a different thread than the one on which the observer was registered.

**Availability**
Available in iOS 4.3 and later.

**Declared In**
`AVPlayerItem.h`

# AVPlayerItemAccessLog Class Reference

| | |
|---|---|
| **Inherits from** | NSObject |
| **Conforms to** | NSCopying<br>NSObject (NSObject) |
| **Framework** | /System/Library/Frameworks/AVFoundation.framework |
| **Availability** | Available in iOS 4.3 and later. |
| **Declared in** | AVPlayerItem.h |

## Overview

You use an `AVPlayerItemAccessLog` object to retrieve the access log associated with an `AVPlayerItem` object.

An `AVPlayerItemAccessLog` object accumulates key metrics about network playback and presents them as a collection of `AVPlayerItemAccessLogEvent` instances. Each event instance collates the data that relates to each uninterrupted period of playback.

## Tasks

### Accessing Log Data

`events` (page 312) *property*
A chronologically ordered array of `AVPlayerItemAccessLogEvent` objects. (read-only)

- `extendedLogData` (page 312)
Returns a serialized representation of the access log in the Extended Log File Format.

- `extendedLogDataStringEncoding` (page 312)
Returns the string ecoding of the extended log data.

## Properties

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

### events

A chronologically ordered array of `AVPlayerItemAccessLogEvent` objects. (read-only)

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

**Discussion**
The array contains `AVPlayerItemAccessLogEvent` objects that represent the chronological sequence of events contained in the access log.

This property is not observable using key-value observing.

**Availability**
Available in iOS 4.3 and later.

**Declared In**
`AVPlayerItem.h`

## Instance Methods

### extendedLogData

Returns a serialized representation of the access log in the Extended Log File Format.

```
- (NSData *)extendedLogData
```

**Return Value**
A serialized representation of the access log in the Extended Log File Format.

**Discussion**
This method converts the web server access log into a textual format that conforms to the W3C Extended Log File Format for web server log files. For more information, see http://www.w3.org/pub/WWW/TR/WD-logfile.html.

You can generate a string suitable for console output using:

```
[[NSString alloc] initWithData:[myLog extendedLogData] encoding:[myLog
extendedLogDataStringEncoding]]
```

**Availability**
Available in iOS 4.3 and later.

**See Also**
`- extendedLogDataStringEncoding` (page 312)

**Declared In**
`AVPlayerItem.h`

### extendedLogDataStringEncoding

Returns the string ecoding of the extended log data.

```
- (NSStringEncoding)extendedLogDataStringEncoding
```

**Return Value**
The string encoding of the data returned by `extendedLogData` (page 312).

**Availability**
Available in iOS 4.3 and later.

**See Also**
- `extendedLogData` (page 312)

**Declared In**
`AVPlayerItem.h`

# AVPlayerItemAccessLogEvent Class Reference

| | |
|---|---|
| **Inherits from** | NSObject |
| **Conforms to** | NSCopying<br>NSObject (NSObject) |
| **Framework** | /System/Library/Frameworks/AVFoundation.framework |
| **Availability** | Available in iOS 4.3 and later. |
| **Declared in** | AVPlayerItem.h |

## Overview

An `AVPlayerItemAccessLogEvent` object represents a single item in an `AVPlayerItem` object's access log.

An `AVPlayerItemAccessLog` object provides named properties for accessing the data fields of each log event. None of the properties of this class are observable using key-value observing.

## Tasks

### Data Properties

`numberOfSegmentsDownloaded` (page 317) *property*
A count of the media segments downloaded from the server to this client. (read-only)

`playbackStartDate` (page 319) *property*
The date and time at which playback began for this event. (read-only)

`URI` (page 320) *property*
The URI of the playback item (read-only)

`serverAddress` (page 320) *property*
The IP address of the server that was the source of the last delivered media segment. (read-only)

`numberOfServerAddressChanges` (page 318) *property*
A count of changes to the server address over the last uninterrupted period of playback. (read-only)

`playbackSessionID` (page 319) *property*
A GUID that identifies the playback session. (read-only)

`playbackStartOffset` (page 319) *property*
An offset into the playlist where the last uninterrupted period of playback began, in seconds (read-only)

`segmentsDownloadedDuration` (page 320) *property*
The accumulated duration of the media downloaded, in seconds. (read-only)

`durationWatched` (page 316) *property*
The accumulated duration of the media played, in seconds. (read-only)

`numberOfStalls` (page 318) *property*
The total number of playback stalls encountered. (read-only)

`numberOfBytesTransferred` (page 317) *property*
The accumulated number of bytes transferred by the item. (read-only)

`indicatedBitrate` (page 316) *property*
The throughput required to play the stream, as advertised by the server, in bits per second. (read-only)

`observedBitrate` (page 318) *property*
The empirical throughput across all media downloaded, in bits per second. (read-only)

`numberOfDroppedVideoFrames` (page 317) *property*
The total number of dropped video frames (read-only)

# Properties

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

## durationWatched

The accumulated duration of the media played, in seconds. (read-only)

```
@property(nonatomic, readonly) NSTimeInterval durationWatched
```

**Discussion**
The property corresponds to "c-duration-watched".

The value of this property is negative if unknown.

**Availability**
Available in iOS 4.3 and later.

**Declared In**
`AVPlayerItem.h`

## indicatedBitrate

The throughput required to play the stream, as advertised by the server, in bits per second. (read-only)

```
@property(nonatomic, readonly) double indicatedBitrate
```

**Discussion**
The property corresponds to "sc-indicated-bitrate".

The value of this property is negative if unknown.

**Availability**
Available in iOS 4.3 and later.

**Declared In**
`AVPlayerItem.h`

## numberOfBytesTransferred

The accumulated number of bytes transferred by the item. (read-only)

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

**Discussion**
The property corresponds to “bytes”.

The value of this property is negative if unknown.

**Availability**
Available in iOS 4.3 and later.

**Declared In**
`AVPlayerItem.h`

## numberOfDroppedVideoFrames

The total number of dropped video frames (read-only)

```
@property(nonatomic, readonly) NSInteger numberOfDroppedVideoFrames
```

**Discussion**
The property corresponds to “c-frames-dropped”.

The value of this property is negative if unknown.

**Availability**
Available in iOS 4.3 and later.

**Declared In**
`AVPlayerItem.h`

## numberOfSegmentsDownloaded

A count of the media segments downloaded from the server to this client. (read-only)

```
@property(nonatomic, readonly) NSInteger numberOfSegmentsDownloaded
```

**Discussion**
The property corresponds to “sc-count”.

The value of this property is negative if unknown.

**Availability**
Available in iOS 4.3 and later.

**Declared In**
`AVPlayerItem.h`

## numberOfServerAddressChanges

A count of changes to the server address over the last uninterrupted period of playback. (read-only)

```
@property(nonatomic, readonly) NSInteger numberOfServerAddressChanges
```

**Discussion**
The property corresponds to “s-ip-changes”.

The value of this property is negative if unknown.

**Availability**
Available in iOS 4.3 and later.

**See Also**
`@property serverAddress` (page 320)

**Declared In**
`AVPlayerItem.h`

## numberOfStalls

The total number of playback stalls encountered. (read-only)

```
@property(nonatomic, readonly) NSInteger numberOfStalls
```

**Discussion**
The property corresponds to “c-stalls”.

The value of this property is negative if unknown.

**Availability**
Available in iOS 4.3 and later.

**Declared In**
`AVPlayerItem.h`

## observedBitrate

The empirical throughput across all media downloaded, in bits per second. (read-only)

```
@property(nonatomic, readonly) double observedBitrate
```

**Discussion**
The property corresponds to “c-observed-bitrate”.

The value of this property is negative if unknown.

**Availability**
Available in iOS 4.3 and later.

**Declared In**
`AVPlayerItem.h`

## playbackSessionID

A GUID that identifies the playback session. (read-only)

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

**Discussion**
This value is used in HTTP requests.

The property corresponds to "cs-guid".

The value of this property is `nil` if unknown.

**Availability**
Available in iOS 4.3 and later.

**Declared In**
`AVPlayerItem.h`

## playbackStartDate

The date and time at which playback began for this event. (read-only)

```
@property(nonatomic, readonly) NSDate *playbackStartDate
```

**Discussion**
The property corresponds to "date".

The value of this property is `nil` if unknown.

**Availability**
Available in iOS 4.3 and later.

**Declared In**
`AVPlayerItem.h`

## playbackStartOffset

An offset into the playlist where the last uninterrupted period of playback began, in seconds (read-only)

```
@property(nonatomic, readonly) NSTimeInterval playbackStartOffset
```

**Discussion**
The property corresponds to "c-start-time".

The value of this property is negative if unknown.

**Availability**
Available in iOS 4.3 and later.

**Declared In**
`AVPlayerItem.h`

## segmentsDownloadedDuration

The accumulated duration of the media downloaded, in seconds. (read-only)

```
@property(nonatomic, readonly) NSTimeInterval segmentsDownloadedDuration
```

**Discussion**
The property corresponds to "-duration-downloaded".

The value of this property is negative if unknown.

**Availability**
Available in iOS 4.3 and later.

**Declared In**
`AVPlayerItem.h`

## serverAddress

The IP address of the server that was the source of the last delivered media segment. (read-only)

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

**Discussion**
The property corresponds to "s-ip".

The value of this property is `nil` if unknown.

**Availability**
Available in iOS 4.3 and later.

**See Also**
`@property numberOfServerAddressChanges` (page 318)

**Declared In**
`AVPlayerItem.h`

## URI

The URI of the playback item (read-only)

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

**Discussion**
The property corresponds to "uri".

The value of this property may be `nil` if the URI is unknown.

**Availability**
Available in iOS 4.3 and later.

**Declared In**
`AVPlayerItem.h`

# AVPlayerItemErrorLog Class Reference

| | |
|---|---|
| **Inherits from** | NSObject |
| **Conforms to** | NSCopying<br>NSObject (NSObject) |
| **Framework** | /System/Library/Frameworks/AVFoundation.framework |
| **Availability** | Available in iOS 4.3 and later. |
| **Declared in** | AVPlayerItem.h |

## Overview

You use an `AVPlayerItemErrorLog` object to retrieve the error log associated with an `AVPlayerItem` object.

## Tasks

### Accessing Error Data

`events` (page 324) *property*
A chronologically ordered array of `AVPlayerItemErrorLogEvent` objects. (read-only)

- `extendedLogData` (page 324)
Returns a serialized representation of the error log in the Extended Log File Format.

- `extendedLogDataStringEncoding` (page 324)
Returns the string ecoding of the extended log data.

## Properties

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

### events

A chronologically ordered array of `AVPlayerItemErrorLogEvent` objects. (read-only)

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

**Discussion**
The array contains `AVPlayerItemErrorLogEvent` objects that represent the chronological sequence of events contained in the error log.

This property is not observable using key-value observing.

**Availability**
Available in iOS 4.3 and later.

**Declared In**
`AVPlayerItem.h`

## Instance Methods

### extendedLogData

Returns a serialized representation of the error log in the Extended Log File Format.

```
- (NSData *)extendedLogData
```

**Return Value**
A serialized representation of the error log in the Extended Log File Format.

**Discussion**
This method converts the web server error log into a textual format that conforms to the W3C Extended Log File Format for web server log files. For more information, see http://www.w3.org/pub/WWW/TR/WD-log-file.html.

You can generate a string suitable for console output using:

```
[[NSString alloc] initWithData:[myLog extendedLogData] encoding:[myLog
extendedLogDataStringEncoding]]
```

**Availability**
Available in iOS 4.3 and later.

**See Also**
`- extendedLogDataStringEncoding` (page 324)

**Declared In**
`AVPlayerItem.h`

### extendedLogDataStringEncoding

Returns the string ecoding of the extended log data.

```
- (NSStringEncoding)extendedLogDataStringEncoding
```

**Return Value**
The string encoding of the data returned by `extendedLogData` (page 324).

**Availability**
Available in iOS 4.3 and later.

**See Also**
- `extendedLogData` (page 324)

**Declared In**
`AVPlayerItem.h`

# AVPlayerItemErrorLogEvent Class Reference

| | |
|---|---|
| **Inherits from** | NSObject |
| **Conforms to** | NSCopying<br>NSObject (NSObject) |
| **Framework** | /System/Library/Frameworks/AVFoundation.framework |
| **Availability** | Available in iOS 4.3 and later. |
| **Declared in** | AVPlayerItem.h |

## Overview

An `AVPlayerItemErrorLogEvent` object represents a single item in an `AVPlayerItem` object's error log.

An `AVPlayerItemErrorLogEvent` object provides named properties for accessing the data fields of each log event. None of the properties of this class are observable using key-value observing.

## Tasks

### Information About the Event

`date` (page 328) *property*
The date and time when the error occurred. (read-only)

`URI` (page 330) *property*
The URI of the playback item (read-only)

`serverAddress` (page 329) *property*
The IP address of the server that was the source of the error. (read-only)

`playbackSessionID` (page 329) *property*
A GUID that identifies the playback session. (read-only)

`errorStatusCode` (page 329) *property*
A unique error code identifier. (read-only)

`errorDomain` (page 328) *property*
The domain of the error. (read-only)

`errorComment` (page 328) *property*
A description of the error encountered (read-only)

# Properties

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

## date

The date and time when the error occurred. (read-only)

```
@property(nonatomic, readonly) NSDate *date
```

**Discussion**
The property corresponds to "date".

The value of this property may be `nil` if the date is unknown.

**Availability**
Available in iOS 4.3 and later.

**Declared In**
`AVPlayerItem.h`

## errorComment

A description of the error encountered (read-only)

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

**Discussion**
The property corresponds to "comment".

The value of this property may be `nil` if further information is not available.

**Availability**
Available in iOS 4.3 and later.

**Declared In**
`AVPlayerItem.h`

## errorDomain

The domain of the error. (read-only)

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

**Discussion**
The property corresponds to "domain".

**Availability**
Available in iOS 4.3 and later.

**Declared In**
`AVPlayerItem.h`

## errorStatusCode

A unique error code identifier. (read-only)

```
@property(nonatomic, readonly) NSInteger errorStatusCode
```

**Discussion**
The property corresponds to “status”.

**Availability**
Available in iOS 4.3 and later.

**Declared In**
`AVPlayerItem.h`

## playbackSessionID

A GUID that identifies the playback session. (read-only)

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

**Discussion**
The property corresponds to “cs-guid”.

The value of this property is used in HTTP requests, and may be `nil` if the GUID is unknown.

**Availability**
Available in iOS 4.3 and later.

**Declared In**
`AVPlayerItem.h`

## serverAddress

The IP address of the server that was the source of the error. (read-only)

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

**Discussion**
The property corresponds to “s-ip”.

The value of this property can be either an IPv4 or IPv6 address, and may be `nil` if the address is unknown.

**Availability**
Available in iOS 4.3 and later.

**Declared In**
`AVPlayerItem.h`

## URI

The URI of the playback item (read-only)

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

**Discussion**
The property corresponds to “uri”.

The value of this property may be `nil` if the URI is unknown.

**Availability**
Available in iOS 4.3 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 331) *property*
The asset track for which the player item represents presentation state. (read-only)

`enabled` (page 332) *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)<br>CAMediaTiming (CALayer)<br>NSObject (NSObject) |
| **Framework** | /System/Library/Frameworks/AVFoundation.framework |
| **Availability** | Available in iOS 4.0 and later. |
| **Declared in** | AVPlayerLayer.h |
| **Companion guide** | AV Foundation Programming Guide |

## Overview

`AVPlayerLayer` is a subclass of `CALayer` to which an `AVPlayer` object can direct its visual output.

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 `videoGravity` (page 335) property defines how the video content is displayed within the player layer's bounds rect.

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.

You can create arbitrary numbers of player layers with the same `AVPlayer` object. Only the most-recently-created player layer will actually display the video content on-screen.

# Tasks

## Miscellaneous

`player` (page 334) *property*
The player for which the player layer displays visual output.

`+ playerLayerWithPlayer:` (page 335)
Returns a player layer to display the visual output of a specified player.

`readyForDisplay` (page 334) *property*
Indicates whether the first video frame has been made ready for display for the current item of the associated player. (read-only)

`videoGravity` (page 335) *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 first video frame has been made ready for display for the current item of the associated player. (read-only)

```
@property(nonatomic, readonly, getter=isReadyForDisplay) BOOL readyForDisplay
```

**Discussion**
Use this property as an indicator of when best to show or animate-in a player layer into view. An player layer may be displayed, or made visible, while this property is `NO`, however the layer will not have any user-visible content until the value becomes `YES`.

This property remains `NO` for a player's `currentItem` whose asset contains no enabled video tracks.

**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**
Options are `AVLayerVideoGravityResizeAspect`, `AVLayerVideoGravityResizeAspectFill`, and `AVLayerVideoGravityResize`. The default is `player` (page 334).

This property is animatable.

**Availability**
Available in iOS 4.0 and later.

**See Also**
`bounds` (`CALayer`)

**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`

# AVQueuePlayer Class Reference

| | |
|---|---|
| **Inherits from** | AVPlayer : NSObject |
| **Conforms to** | NSObject (NSObject) |
| **Framework** | /System/Library/Frameworks/AVFoundation.framework |
| **Availability** | Available in iOS 4.1 and later. |
| **Declared in** | AVPlayer.h |

## Overview

`AVQueuePlayer` is a subclass of `AVPlayer` you use to play a number of items in sequence.

## Tasks

### Creating a Queue Player

`- initWithItems:` (page 339)
Initializes an instance of `AVQueuePlayer` by enqueueing the player items from a given array.

`+ queuePlayerWithItems:` (page 338)
Returns an instance of `AVQueuePlayer` initialized to play items from a given array.

### Managing Items

`- advanceToNextItem` (page 338)
Ends playback of the current item and initiates playback of the next item in the player's queue.

`- canInsertItem:afterItem:` (page 339)
Returns a Boolean value that indicates whether a given player item can be inserted into the player's queue.

`- insertItem:afterItem:` (page 340)
Places given player item after a specified item in the queue.

- `items` (page 340)
  Returns an array of the currently enqueued items.
- `removeAllItems` (page 340)
  Removes all the items from the queue.
- `removeItem:` (page 341)
  Removes a given player item from the queue.

# Class Methods

## queuePlayerWithItems:

Returns an instance of `AVQueuePlayer` initialized to play items from a given array.

```
+ (AVQueuePlayer *)queuePlayerWithItems:(NSArray *)items
```

**Parameters**

*items*
  An array of `AVPlayerItem` objects with which initially to populate the player's queue.

**Return Value**
An instance of `AVQueuePlayer` initialized to play the player items in *items*.

**Discussion**

**Availability**
Available in iOS 4.1 and later.

**See Also**
- `initWithItems:` (page 339)
- `insertItem:afterItem:` (page 340)

**Declared In**
`AVPlayer.h`

# Instance Methods

## advanceToNextItem

Ends playback of the current item and initiates playback of the next item in the player's queue.

```
- (void)advanceToNextItem
```

**Discussion**
This method also removes the current item from the play queue.

**Availability**
Available in iOS 4.1 and later.

**Declared In**
AVPlayer.h

## canInsertItem:afterItem:

Returns a Boolean value that indicates whether a given player item can be inserted into the player's queue.

```
- (BOOL)canInsertItem:(AVPlayerItem *)item afterItem:(AVPlayerItem *)afterItem
```

**Parameters**

*item*
The AVPlayerItem object to test.

*afterItem*
The item that *item* is to follow in the queue. Pass nil to test whether *item* can be appended to the queue.

**Return Value**
YES if *item* can be appended to the queue, otherwise NO.

**Discussion**
Adding the same item to a player at more than one position in the queue is not supported.

**Availability**
Available in iOS 4.1 and later.

**See Also**
- insertItem:afterItem: (page 340)

**Declared In**
AVPlayer.h

## initWithItems:

Initializes an instance of AVQueuePlayer by enqueueing the player items from a given array.

```
- (id)initWithItems:(NSArray *)items
```

**Parameters**

*items*
An array of AVPlayerItem objects with which initially to populate the player's queue.

**Return Value**
An instance of AVQueuePlayer initialized to play the player items in *items*.

**Discussion**

**Availability**
Available in iOS 4.1 and later.

**See Also**
+ queuePlayerWithItems: (page 338)
- insertItem:afterItem: (page 340)

**Declared In**
AVPlayer.h

## insertItem:afterItem:

Places given player item after a specified item in the queue.

```
- (void)insertItem:(AVPlayerItem *)item afterItem:(AVPlayerItem *)afterItem
```

**Parameters**

*item*
The item to be inserted.

*afterItem*
The item that the newly inserted item should follow in the queue. Pass `nil` to append the item to the queue.

**Discussion**

**Availability**
Available in iOS 4.1 and later.

**See Also**
- `canInsertItem:afterItem:` (page 339)

**Declared In**
AVPlayer.h

## items

Returns an array of the currently enqueued items.

```
- (NSArray *)items
```

**Return Value**
An array of the currently enqueued items

**Discussion**
The array contains `AVPlayerItem` objects

**Availability**
Available in iOS 4.1 and later.

**Declared In**
AVPlayer.h

## removeAllItems

Removes all the items from the queue.

```
- (void)removeAllItems
```

**Discussion**
This has the side-effect of stopping playback by the player.

**Availability**
Available in iOS 4.1 and later.

**Declared In**
`AVPlayer.h`

## removeItem:

Removes a given player item from the queue.

```
- (void)removeItem:(AVPlayerItem *)item
```

**Parameters**

*item*

The item to be removed.

**Discussion**
If *item* is currently playing, this has the same effect as `advanceToNextItem` (page 338).

**Availability**
Available in iOS 4.1 and later.

**Declared In**
`AVPlayer.h`

# AVSynchronizedLayer Class Reference

| | |
|---|---|
| **Inherits from** | CALayer : NSObject |
| **Conforms to** | NSCoding (CALayer)<br>CAMediaTiming (CALayer)<br>NSObject (NSObject) |
| **Framework** | /System/Library/Frameworks/AVFoundation.framework |
| **Availability** | Available in iOS 4.0 and later. |
| **Declared in** | AVSynchronizedLayer.h |
| **Companion guide** | AV Foundation Programming Guide |

## 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.

A synchronized layers is similar to a `CATransformLayer` object in that it doesn't display anything itself, it just 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 344)
Returns a new synchronized layer with timing synchronized with a given player item.

## Managing the Player Item

`playerItem` (page 344) *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

# AVTimedMetadataGroup Class Reference

| | |
|---|---|
| **Inherits from** | NSObject |
| **Conforms to** | NSCopying<br>NSObject (NSObject) |
| **Framework** | /System/Library/Frameworks/AVFoundation.framework |
| **Availability** | Available in iOS 4.3 and later. |
| **Declared in** | AVTimedMetadataGroup.h |

## Overview

You use an `AVTimedMetadataGroup` object to represent a collection of metadata items.

AV Foundation also provides a mutable subclass, `AVMutableTimedMetadataGroup`, that you can use to create your own collections.

## Tasks

### Creating and Analyzing a Metadata Group

`- initWithItems:timeRange:` (page 348)
Returns a metadata group initialized with given metadata items.

`timeRange` (page 348) *property*
The time range of the metadata. (read-only)

`items` (page 348) *property*
The metadata items in the group. (read-only)

## Properties

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

### items

The metadata items in the group. (read-only)

```
@property(readonly, copy) NSArray *items
```

**Discussion**
The array contains instances of `AVMetadataItem`.

**Availability**
Available in iOS 4.3 and later.

**Declared In**
`AVTimedMetadataGroup.h`

### timeRange

The time range of the metadata. (read-only)

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

**Discussion**

**Availability**
Available in iOS 4.3 and later.

**Declared In**
`AVTimedMetadataGroup.h`

## Instance Methods

### initWithItems:timeRange:

Returns a metadata group initialized with given metadata items.

```
- (id)initWithItems:(NSArray *)items timeRange:(CMTimeRange)timeRange
```

**Parameters**

*items*
An array of `AVMetadataItem` objects.

*timeRange*
The time range of the metadata contained in *items*.

**Return Value**
A metadata group initialized with *items*.

**Discussion**

**Availability**
Available in iOS 4.3 and later.

**Declared In**
`AVTimedMetadataGroup.h`

# AVURLAsset Class Reference

| | |
|---|---|
| **Inherits from** | AVAsset : NSObject |
| **Conforms to** | NSCopying (AVAsset)<br>AVAsynchronousKeyValueLoading (AVAsset)<br>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 353)
Initializes an asset for inspection of a resource referenced by a given URL.

`+ URLAssetWithURL:options:` (page 352)
Returns an asset for inspection of a resource referenced by a given URL.

### Accessing the URL

`URL` (page 352) *property*
The URL with which the asset was initialized. (read-only)

### Finding Compatible Tracks

`- compatibleTrackForCompositionTrack:` (page 353)
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 353)
`+ URLAssetWithURL:options:` (page 352)

**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 354).

**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 353)
`@property URL` (page 352)

**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 257) (`AVMutableCompositionTrack`). This method is the logical complement of `mutableTrackCompatibleWithTrack:` (page 249).

**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 354).

**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 352)
`@property URL` (page 352)

**Declared In**
`AVAsset.h`

# Constants

## Initialization Options

Keys for options dictionary for use with `initWithURL:options:` (page 353) and `URLAssetWithURL:options:` (page 352).

```
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<br>NSMutableCopying<br>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 356) *property*
The interval for which the video composition should render composed video frames. (read-only)

`renderSize` (page 357) *property*
The size at which the video composition should render. (read-only)

`instructions` (page 356) *property*
The video composition instructions. (read-only)

`animationTool` (page 356) *property*
A video composition tool to use with Core Animation in offline rendering. (read-only)

`renderScale` (page 357) *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<br>NSCopying<br>NSMutableCopying<br>NSObject (NSObject) |
| **Framework** | /System/Library/Frameworks/AVFoundation.framework |
| **Availability** | Available in iOS 4.0 and later. |
| **Declared in** | AVVideoComposition.h |
| **Companion guide** | AV Foundation Programming Guide |

## 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 360) *property*
The background color of the composition.

`layerInstructions` (page 360) *property*
An array of instances of `AVVideoCompositionLayerInstruction` that specify how video frames from source tracks should be layered and composed. (read-only)

`timeRange` (page 361) *property*
The time range during which the instruction is effective. (read-only)

`enablePostProcessing` (page 360) *property*
Indicates whether post processing is required for the video composition instruction. (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 supported colors 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 `NULL`, the video compositor uses a default background color of opaque black.

**Availability**
Available in iOS 4.0 and later.

**Declared In**
`AVVideoComposition.h`

## enablePostProcessing

Indicates whether post processing is required for the video composition instruction. (read-only)

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

**Discussion**
A value of `NO` indicates that no post processing is required for the whole duration of the video composition instruction. The composition process is more efficient if the value is `NO`.

The value is `YES` by default.

**Availability**
Available in iOS 4.0 and later.

**See Also**
`enablePostProcessing` (page 272)

**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 the property value is `nil`, the output is a fill of the background color.

**Availability**
Available in iOS 4.0 and later.

**See Also**
`@property backgroundColor` (page 360)

**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.

**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 365)
  Encodes a given `CMTime` structure and associates it with a specified key.
- `encodeCMTimeRange:forKey:` (page 366)
  Encodes a given `CMTimeRange` structure and associates it with a specified key.
- `encodeCMTimeMapping:forKey:` (page 365)
  Encodes a given `CMTimeMapping` structure and associates it with a specified key.

### Decoding Core Media Time Structures

- `decodeCMTimeForKey:` (page 364)
  Returns the `CMTime` structure associated with a given key.
- `decodeCMTimeRangeForKey:` (page 364)
  Returns the `CMTimeRange` structure associated with a given key.
- `decodeCMTimeMappingForKey:` (page 364)
  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 365)

**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 365)

**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 366)

**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 364)

**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 364)

**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 364)

**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 368)
Returns a value object that contains a given `CMTime` structure.

`+ valueWithCMTimeMapping:` (page 368)
Returns a value object that contains a given `CMTimeMapping` structure.

`+ valueWithCMTimeRange:` (page 368)
Returns a value object that contains a given `CMTimeRange` structure.

### Retrieving Core Media Time Structures

`- CMTimeMappingValue` (page 369)
Returns a `CMTimeMapping` structure representation of the receiver.

`- CMTimeRangeValue` (page 369)
Returns a `CMTimeRange` structure representation of the receiver.

`- CMTimeValue` (page 370)
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 370)

**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 369)

**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 369)

**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 368)

**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 368)

**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 368)

**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 375).
- 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 374).

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 374).

# Tasks

## Protocol Methods

- `loadValuesAsynchronouslyForKeys:completionHandler:` (page 374)
  Tells the asset to load the values of any of the specified keys that are not already loaded. (required)
- `statusOfValueForKey:error:` (page 375)
  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 27) 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 375)

**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 376), 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 374).

**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 374) 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 374)

**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 376).

**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 375).

# 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<br>AQOfflineRenderTest<br>iPhoneExtAudioFileConvertTest |

## 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 379)
Called when a sound has finished playing.

### Responding to an Audio Decoding Error

`- audioPlayerDecodeErrorDidOccur:error:` (page 378)
Called when an audio player encounters a decoding error during playback.

### Handling Audio Interruptions

- `audioPlayerBeginInterruption:` (page 378)
  Called when an audio player is interrupted, such as by an incoming phone call.
- `audioPlayerEndInterruption:` (page 379)
  Called after your audio session interruption ends.
- `audioPlayerEndInterruption:withFlags:` (page 380)
  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 380)

**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 135) method.

**Availability**
Available in iOS 2.2 and later.

**See Also**
- audioPlayerBeginInterruption: (page 378)
- audioPlayerEndInterruption:withFlags: (page 380)

**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.

**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 135) method.

If this delegate method receives the AVAudioSessionInterruptionFlags_ShouldResume 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 379) method.

**Availability**
Available in iOS 4.0 and later.

**See Also**
- audioPlayerBeginInterruption: (page 378)

**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 382)
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 383)
Called when an audio recorder encounters an encoding error during recording.

### Handling Audio Interruptions

`- audioRecorderBeginInterruption:` (page 382)
Called when the audio session is interrupted during a recording, such as by an incoming phone call.

- `audioRecorderEndInterruption:` (page 383)
  Called after your audio session interruption ends.
- `audioRecorderEndInterruption:withFlags:` (page 384)
  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 384)

**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 145) method.

**Availability**
Available in iOS 3.0 and later.

**See Also**
- `audioRecorderBeginInterruption:` (page 382)
- `audioRecorderEndInterruption:withFlags:` (page 384)

**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`.

### 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 145) method.

If this delegate method receives the `AVAudioSessionInterruptionFlags_ShouldResume` 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 383) method.

### Availability

Available in iOS 4.0 and later.

### See Also

- `audioRecorderBeginInterruption:` (page 382)

### Declared In

`AVAudioRecorder.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 385)
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 388)
  Called when the capture object starts saving data to a file.
- `captureOutput:didFinishRecordingToOutputFileAtURL:fromConnections:error:` (page 387)
  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`.

## Linear PCM Format Defines

Audio setting defines that apply to linear PCM audio formats.

```
#define AVLinearPCMIsNonInterleavedKey AVLinearPCMIsNonInterleaved
```

**Constants**

`AVLinearPCMIsNonInterleavedKey`

See `AVLinearPCMIsNonInterleaved` (page 396).

Available in iOS 4.1 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 398).

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 398).

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 397) 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<br>AVAnimation.h<br>AVMediaFormat.h |
| **Companion guide** | AV Foundation Programming Guide |

## 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 AVFileTypeQuickTimeMovie;
NSString *const AVFileTypeMPEG4;
NSString *const AVFileTypeAppleM4V;
NSString *const AVFileTypeAppleM4A;
NSString *const AVFileType3GPP;
NSString *const AVFileTypeCoreAudioFormat;
NSString *const AVFileTypeWAVE;
NSString *const AVFileTypeAIFF;
NSString *const AVFileTypeAIFC;
NSString *const AVFileTypeAMR;
```

### Constants

`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`.

`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`.

`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`.

`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`.

`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`.

`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`.

`AVFileTypeWAVE`
: A UTI for the WAVE audio 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`.

`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`.

`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`.

`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;
NSString *const AVErrorMediaTypeKey;
NSString *const AVErrorMediaSubTypeKey;
```

**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`.

`AVErrorMediaTypeKey`

The corresponding value is an `NSString` object that specified a media format.

Possible values are given in AVMediaFormat.h.

Available in iOS 4.3 and later.

Declared in `AVError.h`.

`AVErrorMediaSubTypeKey`

The corresponding value is an array of `NSNumber` objects that specify media subtypes.

The types are represented by four character codes (4ccs), as defined in CoreAudioTypes.h for audio media and in CMFormatDescription.h for video media.

Available in iOS 4.3 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,
    AVErrorDecoderNotFound                                   = -11833,
    AVErrorEncoderNotFound                                   = -11834,
    AVErrorContentIsNotAuthorized                            = -11835,
};
```

### 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`.

`AVErrorDecoderNotFound`
The decoder for the given media was not found

The error's `userInfo` may contain values for the keys `AVErrorMediaTypeKey` (page 412) and `AVErrorMediaSubTypeKey` (page 412), if they are available.

Available in iOS 4.3 and later.

Declared in `AVError.h`.

`AVErrorEncoderNotFound`

The requested encoder was not found.

The error's `userInfo` may contain values for the keys `AVErrorMediaTypeKey` (page 412) and `AVErrorMediaSubTypeKey` (page 412), if they are available.

Available in iOS 4.3 and later.

Declared in `AVError.h`.

`AVErrorContentIsNotAuthorized`

The user is not authorized to play the media.

Available in iOS 4.3 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 AVMetadataQuickTimeMetadataKeyTitle;
NSString *const AVMetadataQuickTimeMetadataKeyCollectionUser;
NSString *const AVMetadataQuickTimeMetadataKeyRatingUser;
NSString *const AVMetadataQuickTimeMetadataKeyLocationName;
NSString *const AVMetadataQuickTimeMetadataKeyLocationBody;
NSString *const AVMetadataQuickTimeMetadataKeyLocationNote;
NSString *const AVMetadataQuickTimeMetadataKeyLocationRole;
NSString *const AVMetadataQuickTimeMetadataKeyLocationDate;
NSString *const AVMetadataQuickTimeMetadataKeyDirectionFacing;
NSString *const AVMetadataQuickTimeMetadataKeyDirectionMotion;
```

```
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`.

`AVMetadataQuickTimeMetadataKeyTitle`

Available in iOS 4.3 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeMetadataKeyCollectionUser`

Available in iOS 4.3 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeMetadataKeyRatingUser`

Available in iOS 4.3 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeMetadataKeyLocationName`

Available in iOS 4.3 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeMetadataKeyLocationBody`

Available in iOS 4.3 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeMetadataKeyLocationNote`

Available in iOS 4.3 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeMetadataKeyLocationRole`

Available in iOS 4.3 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeMetadataKeyLocationDate`

Available in iOS 4.3 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeMetadataKeyDirectionFacing`

Available in iOS 4.3 and later.

Declared in `AVMetadataFormat.h`.

`AVMetadataQuickTimeMetadataKeyDirectionMotion`

Available in iOS 4.3 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 |
| --- | --- |
| 2011-01-06 | Updated for iOS 4.3. |
| 2010-11-15 | Updated for iOS v4.1. |
| 2010-07-23 | Updated for iOS v4.1. |
| 2010-07-13 | Corrected minor typographical error. |
| 2010-05-15 | Updated for iOS 4.0. |
| 2009-03-02 | Updated for iOS 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