QuickTime 7 Update Guide

QuickTime

2005-04-29

Contents

Figures, Tables, and Listings

Introduction to QuickTime 7

QuickTime is the industry standard for multimedia programming and application development, with a rich and evolving API comprised of more than 2500 function calls. Its component-based architecture is highly extensible, enabling applications to display, import, export, modify, and capture a broad range of digital media, including audio, video, still images, text, Flash, MIDI, sprites, VR panoramas, among other media types. QuickTime is designed from the ground up to work with local disk-based media, media accessed over a network, or streams of real-time data.

This document provides detailed information about the new features, changes, and enhanced capabilities that are available in QuickTime 7 for Mac OS X version 10.4.

Who Needs To Read This Document

If you are a QuickTime API-level developer, content author, multimedia producer, or Webmaster who is currently working with QuickTime, you should read this document.

The document is written both for developers who use QuickTime on the Mac OS X platform and want to learn the new programming features of QuickTime 7, and for beginning or experienced Cocoa programmers interested in using QuickTime in their application development.

How This Document Is Organized

This update guide is intended to provide QuickTime developers, as well as other developers new to the platform, with a comprehensive description of the changes and enhancements in this major software release. Beyond this brief introductory chapter, the material discussed in Chapter 2 of the guide points to and cross-references in Chapter 3 the many new functions available in QuickTime 7, with an emphasis on understanding their usage for application developers.

- Chapter 1, "Introduction to QuickTime 7" (page 7), discusses who should read this document, as well as other sources of information about the QuickTime documentation suite.
- Chapter 2, "What's New in QuickTime 7" (page 9), describes in detail the many new and enhanced features available in QuickTime 7. It is intended to provide developers with a conceptual overview, in addition code samples and illustrations of usage, so that developers can take advantage of many of new features in QuickTime 7 in their applications.
- Chapter 3, "New Functions, Data Types, and Constants in QuickTime 7" (page 63), describes all the new QuickTime functions, data structures, constants, and callbacks in this software release.

See Also

For developers who want to take advantage of QuickTime features and functionality, the complete suite of documentation that describes the QuickTime API is available online in HTML and PDF at the QuickTime Reference Library website.

The reference information currently presented in Chapter 3 of this update guide is also available in the QuickTime API Reference document. All of the new QuickTime functions, data structures, and callbacks in QuickTime 7 are incorporated into the QuickTime API Reference for easy access and reference, either in HTML or PDF formats.

If you are new to QuickTime, you should begin by referring to Getting Started With QuickTime, which describes the various starting points and learning paths for working with this rich, multimedia API.

Updates to the QuickTime technical documentation website are provided on a regular basis. Developers can also subscribe to various mailing lists for the latest news and information.

To sign up for any of Apple's Developer Programs, go to: http://developer.apple.com/membership/index.html.

What's New in QuickTime 7

This chapter describes in detail the many new and enhanced features available in QuickTime 7. It is intended to provide developers with a conceptual overview, in addition code samples and illustrations of usage, so that developers can take advantage of many of these new features in QuickTime 7 in their applications.

The new functions discussed in this chapter are cross-referenced, with links, to their complete descriptions in Chapter 3, "New Functions, Data Types, and Constants in QuickTime 7" (page 63).

If you are a QuickTime API-level developer, content author, multimedia producer, or Webmaster who is currently working with QuickTime, you should read this chapter in order to understand the fundamental changes that have taken place in the QuickTime software architecture.

Installing QuickTime 7

QuickTime 7 is installed automatically as part of Mac OS X v10.4.

Hardware and Software Requirements

QuickTime 7 requires the following minimum configuration:

- Mac OS X v10.4, v10.3, or Windows
- PowerPC G3 or better running at 400 MHz or higher
- At least 256 MB of RAM

New Pro Key Required

QuickTime 7 replaces existing point releases of QuickTime 6 for Mac OS X. A new Pro key is required; QuickTime 6 Pro keys will not unlock the Pro features of QuickTime 7.

QuickTime in Perspective

The QuickTime API is dedicated to extending the reach of application developers by letting them invoke the full range of multimedia's capabilities. It supports a wide range of standards-based formats, in addition to proprietary formats from Apple and others. The QuickTime API is not static, however, and has evolved over the course of the last decade to adopt new idioms, new data structures, and new ways of doing things.

The C/C++ portion of the QuickTime API comprises more than 2500 functions that provide services to applications. These services include audio and video capture and playback; movie editing, composition, and streaming; still image import, export, and display; audio-visual interactivity, and more.

A new Cocoa (Objective-C) API for QuickTime, available in Mac OS X v10.4 and v10.3, provides a much less complex programmer interface, and represents a distillation and abstraction of the most essential QuickTime functions as a small set of classes and methods. A great deal of functionality has been packed into a relatively small objective API.

New Features of QuickTime 7

This release of QuickTime includes a number of major new features for users, developers, and content creators, including improvements in the QuickTime architecture, file format, user interface, and API. There are significant improvements in the audio, video, and metadata capabilities, as well as a new Cocoa API, and numerous other enhancements.

- "Changes to QuickTime Player and QuickTime Pro" (page 12) describes the new user interface for QuickTime Player and QuickTime Pro and some of the changes from previous versions of the player.
- "New QuickTime Kit Framework" (page 27) describes a new Cocoa (Objective-C) framework for developing QuickTime applications. This new API opens the world of QuickTime programming to a new group of developers without requiring them to learn the large, complex C/C++ QuickTime API. The new framework encapsulates a tremendous amount of QuickTime functionality in a small, easily-mastered API with a handful of new objects, classes, and methods.
- "Audio Enhancements" (page 29) describes the many new audio features of QuickTime 7, including support for multichannel sound, playback, capture, compression, and export of high-resolution audio, a new sound description, and new functions for movie audio control, audio conversion configuration, audio extraction, movie export, and level and frequency metering.
- "Video Enhancements" (page 45) describes QuickTime's new support for frame reordering video compression and the H.264 codec. Frame reordering support is a major advance that involves new sample tables for video, allowing video frames to have independent decode and display times. This allows improved display, editing, and compression of H.264 and other advanced video codecs. A new set of functions and structures are introduced to allow developers to work with samples that have independent decode and display times.
- "New Abstractions Layers For OpenGL Rendering" (page 49) describes the new Visual Context, an abstraction layer that eliminates dependence on graphics worlds (GWorlds) and supports rendering directly to engines such as OpenGL.
- "Replacing NewMovieFrom... Functions" (page 52) describes the `NewMovieFromProperties` function, which allows you to set up properties before creating a movie. This function also allows you to create movies that are not necessarily associated with a graphics world, movies that can render their output to a visual context, such as an OpenGL texture buffer, and movies that play to a particular audio device.
- "QuickTime Metadata Enhancements and API" (page 53) describes the new QuickTime extensible metadata format, allowing developers to efficiently reference text, audio, video, or other material that describes a movie, a track, or a media. Support is also added for including metadata from other file types in native format; the QuickTime 7 release includes native support for iTunes metadata.
- "QuickTime Sample Table API" (page 56) describes the new API for working with QT Sample Tables, a logical replacement for arrays of media sample references. The new API greatly extends the functionality of media sample references, and the new API supports frame reordering compressed media.

- "JavaScript Support and Accessibility in Safari" (page 58) describes the JavaScript support for the Safari browser. This means you can now use JavaScript to control QuickTime when web pages are viewed using Safari.
- "Other Changes and Enhancements" (page 58) discusses QuickTime 7's new persistent cache option, which is important for web authors and content developers to understand because it may impact the way that QuickTime content is downloaded and saved from their websites. New updates and fixes to QuickTime for Java are also discussed in this section.

New Directions in QuickTime 7

Key areas of change evident in QuickTime 7 are:

- A shift of emphasis toward a Core Audio approach to sound, and away from the Sound Manager approach, throughout QuickTime.
- A shift of emphasis toward configuring components using component properties and an abstraction layer, or context, and away from the exclusive use of standard dialogs supplemented by direct access to low-level components.
- A shift of emphasis toward a more object-oriented organization, with more high-level functionality in QuickTime itself supporting lighter-weight applications.

What Developers Need To Do

If you work with audio at a relatively low level, you should become familiar with the Mac OS X Core Audio framework and learn how it differs from the older Sound Manager. The use of Core Audio concepts and data structures is becoming ubiquitous in QuickTime for both Mac OS X and Windows. For details, see Apple's Core Audio documentation.

If you work directly with components, you should become familiar with the API for discovering, getting, and setting component properties. While standard dialogs for configuration are still common, there are often times when either no dialog or an application-specific dialog is preferable, as well as cases where low-level control or device-specific configuration is needed that a standard dialog cannot supply.

For example, the component property API allows configuration at any level of detail without requiring a user interface dialog or direct communication with low-level components. In many cases, an abstraction layer, or **context**––either visual or audio––can be created, allowing transparent connection to different kinds of low-level components, devices, or rendering engines.

The new extensible QuickTime metadata format, discussed in the section "QuickTime Metadata Enhancements and API" (page 53), uses a similar method of configuration through an abstract set of properties, as a means of "future-proofing" the architecture. The same is true of the new API for working with QuickTime sample tables, described in the section "QuickTime Sample Table API" (page 56).

Object Model Evolution

A substantial reorganization of the QuickTime engine has been taking place "under the hood" in this software release. This reorganization is intended to allow increased access to QuickTime functionality from object-oriented frameworks such as Cocoa (Objective-C).

As the QuickTime document object model continues to evolve, the goal is to provide developers with easier access to the more powerful parts of the QuickTime engine using relatively lightweight object-oriented applications or even scripts––without having to delve into the large and sometimes complex procedural C/C++ QuickTime API. If you haven't experimented with Cocoa and the Xcode tools yet, this is a good time to get started.

In Summary QuickTime 6 through QuickTime 7

The following table summarizes the point releases of QuickTime 6 and the features of QuickTime 7.

QuickTime version	Mac OS X	Windows	Mac OS 9	Features
6	x	x	x	MPEG-4 and lots more.
6.01	x	x	x	Bug fix for QuickTime 6. Last version for all three platforms.
6.03			x	Bug fixes to address security issues. Mac OS 9 only.
6.1	x	x		Improved MPEG-4 video, full-screen modes, wired actions.
6.2	x			Support for iTunes 4, enhanced AAC codec, limited DRM.
6.3	x	x		Improved AAC codec, 3GPP support, which includes AMR codec.
6.4 for Mac OS X	x			New data reference functions, multithreading, new graphics functions, component and movie property access, other API additions.
6.5	x	x		3GPP, 3GPP2, and AMC support for mobile multimedia, Unicode text support.
6.5.1	x	x		Apple Lossless codec for audio.
7	x	x		High-resolution, multichannel audio support, frame reordering video and H.264 support, new Cocoa API, support for rendering to OpenGL and elimination of dependence on graphics worlds (GWorlds), new metadata format, QuickTime sample table API, changes to QuickTime Player and Pro UI.

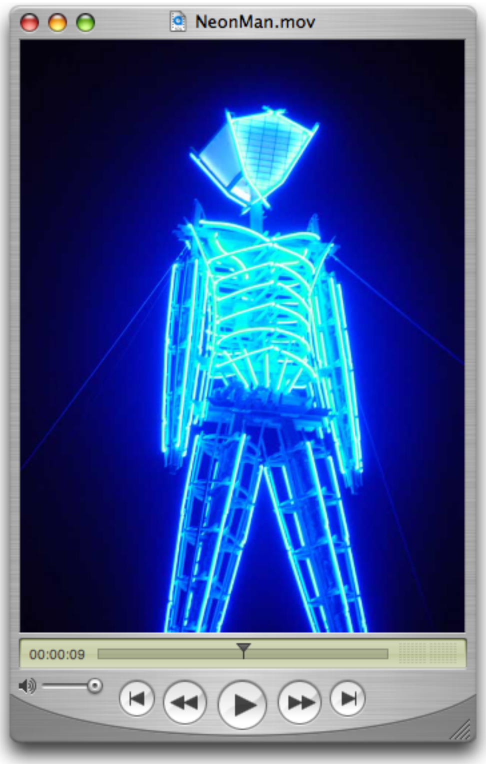
Changes to QuickTime Player and QuickTime Pro

QuickTime 7 introduces a number of new features and changes to the user interface of QuickTime Player and QuickTime Pro. These are briefly described in this section. Both Player and Pro are available in Mac OS X v10.4 and are also backward-compatible with Mac OS X v10.3.

New in QuickTime Player

The new QuickTime Player, shown in Figure 2-1, is a native Cocoa application. The intent of this new design is to better integrate QuickTime Player in general with the Mac OS X user experience.

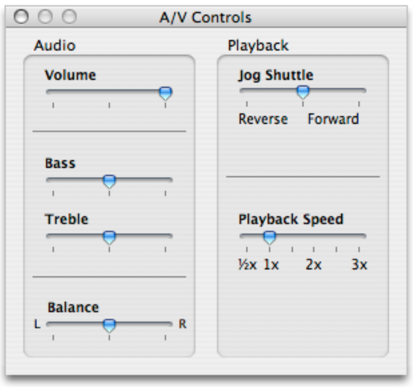
Figure 2-1 The new QuickTime Player application



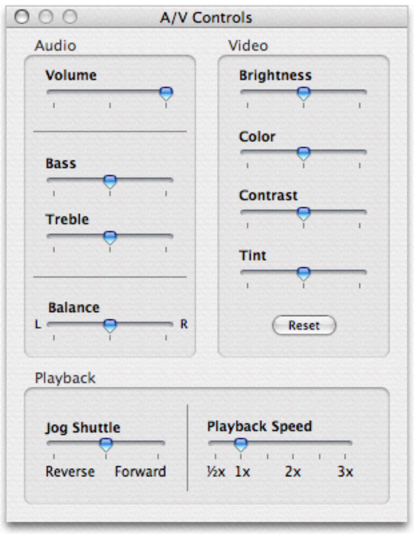
The following are some of the new user-level features available in QuickTime Player:

- **H.264 video support**. This state-of-the-art, standards-based codec delivers exceptional-quality video at the lowest data rate possible, across the entire bandwidth spectrum.
- **New audio and playback controls**. Users can use the new A/V Controls window (previously available only to QuickTime Pro users) to adjust settings for the best audio and playback experience. Users can now easily change settings, including playback speed, volume, bass, treble, and balance, as shown in Figure 2-2.

Figure 2-2 New audio, playback, and video controls in QuickTime Player and QuickTime Player with Core Image support in Mac OSX v10.4



For QuickTime Player



For QuickTime Player + video card and Core Image support in Mac OSX v10.4

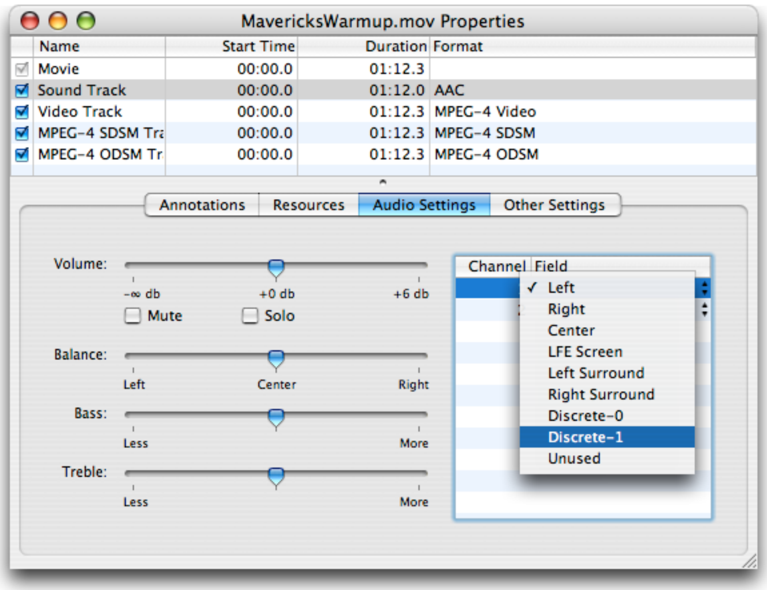
A new video controls panel is also available, as shown in the right portion of Figure 2-2. This option, however, is only available for users with a special video card on Mac OS X v10.4 where Core Image support is provided. The video controls let the user adjust for brightness, color, contrast, and tint.

Note: Core Image extends the basic graphics capabilities of the system to provide a framework for implementing complex visual behaviors in your application. Core Image uses GPU-based acceleration and 32-bit floating-point support to provide fast image processing and pixel-level accurate content.

- **Zero-configuration streaming**. You no longer need to set your Internet connection speed in QuickTime Preferences. QuickTime automatically determines the best connection speed for your computer. If a connection is lost during streaming, QuickTime automatically reconnects to the server.
- **Live resize**. Playback continues smoothly as you change the size of the QuickTime Player window. (Note that there may be hardware dependencies that affect the speed and smoothness of live resizing.)
- **Multichannel audio**. QuickTime Player can now play 24 audio channels––and beyond. With external speakers, you can enjoy the full sound effects of movies and games.

  By accessing the Window > Show Movie Properties dialog and selecting Audio Settings, as shown in Figure 2-3, you can set the volume, balance, bass, and treble for a QuickTime movie. In addition, if you select the sound track property in the dialog, you can set the speaker for each audio channel in that track, specifying the speaker through which the audio can be heard.

Figure 2-3 The audio settings dialog with sliders for audio control and channel speaker assignments



	Name	Start Time	Duration	Format
✓	Movie	00:00.0	01:12.3	
✓	Sound Track	00:00.0	01:12.0	AAC
✓	Video Track	00:00.0	01:12.3	MPEG-4 Video
✓	MPEG-4 SDSM Tra	00:00.0	01:12.3	MPEG-4 SDSM
✓	MPEG-4 ODSM Tr	00:00.0	01:12.3	MPEG-4 ODSM

- **All-new content guide**. The completely redesigned QuickTime Content Guide provides the latest in news, education, and entertainment on the Internet.
- **Screen-reader compatibility**. Using VoiceOver, included with Mac OS X v10.4, users with visual disabilities can enjoy QuickTime Player features.
- **Spotlight-friendly content**. New in Mac OS X v10.4, Spotlight makes it easy to find your QuickTime content. Spotlight can search for movie attributes such as artist, copyright, codec, and so on.
- **Easy access to QuickTime Pro**. Items available only in QuickTime Pro display "Pro" by their names. If you choose one of these items, you'll see a definition of the feature and learn how to upgrade to QuickTime Pro. Note that the designation "Pro" is only present when QuickTime Player is not the Pro version.

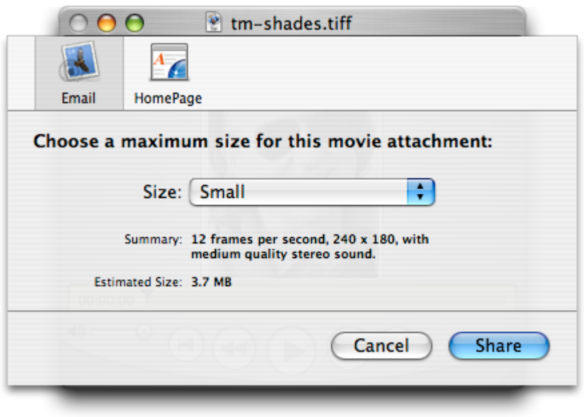
New in QuickTime Pro

The following are some of the new user-level features available in the Pro version of QuickTime Player:

- **Creating H.264 video.** Users can take advantage of this codec for a variety of video needs, ranging from HD (high definition) to 3G (for mobile devices). This new codec provides better quality at lower bandwidth, enabling users to deliver high-quality video over the Internet.
- **Creating multichannel audio**. Users can create a rich multimedia experience by labeling each audio channel (for example, Left, Right, Left Surround, LFE, and so on), as shown in Figure 2-3. QuickTime automatically mixes the audio to work with the speaker setup of each user.
- **Recording audio and video**. With a digital video camera connected to your computer, you can enrich your email messages with video clips. In addition, with enhanced recording of audio and video, users can add narration, for example, to their slide shows.

- **Sharing movies**. Users can easily create a movie file for sending via email or posting to your .Mac HomePage. Select File > Share and a dialog appears that lets you choose a maximum size for the attached movie you want to share and then exports the movie to either your Mail program or to your .Mac HomePage, as shown in Figure 2-4.

Figure 2-4 Sharing attached movies either as email or on your .Mac HomePage



Choose a maximum size for this movie attachment:

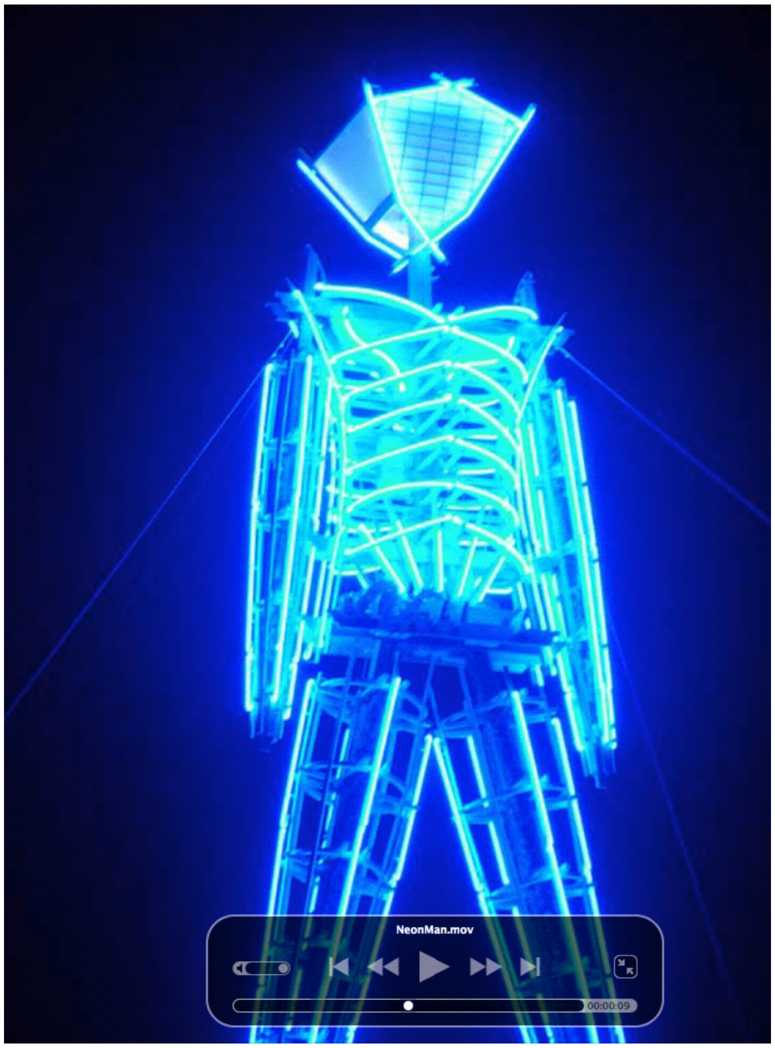
Size: Small

Summary: 12 frames per second, 240 x 180, with medium quality stereo sound.

Estimated Size: 3.7 MB

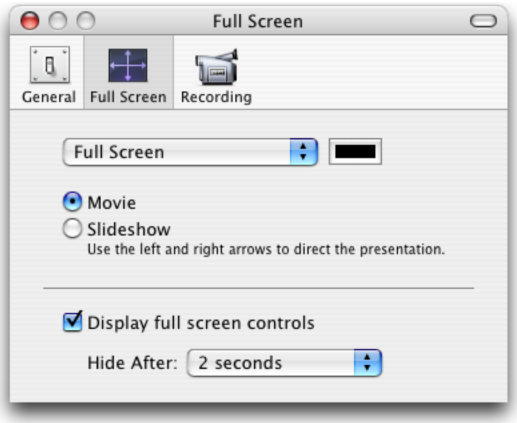
- **Full screen playback enhancements**. Full screen mode now provides floating Dashboard-style controls similar to the controls available for DVD Player. These include pause, play, stop, fast forward, and rewind, as illustrated in Figure 2-5. Users move the pointer to display the controller; after a few seconds, the controller fades away. Note that the controller does not appear with interactive movies when the mouse is moved, so that it does not interfer with movie content. Users can press the keyboard control-C to make it appear or disappear immediately. This new zooming transition, enabling you to go in and out of Full Screen, is dependent on the user's computer hardware, as well as the media being played back.

Figure 2-5 Full screen controls with a floating movie controller



Users can access full screen mode by choosing the View > Full Screen or using its keyboard equivalent. To display the DVD-style full screen controls, users choose QuickTime Player > Preferences > Full Screen and select "Display full screen controls," as shown in Figure 2-6.

Figure 2-6 Full Screen preferences with the full screen controls selected



Full Screen

[✓ Movie]

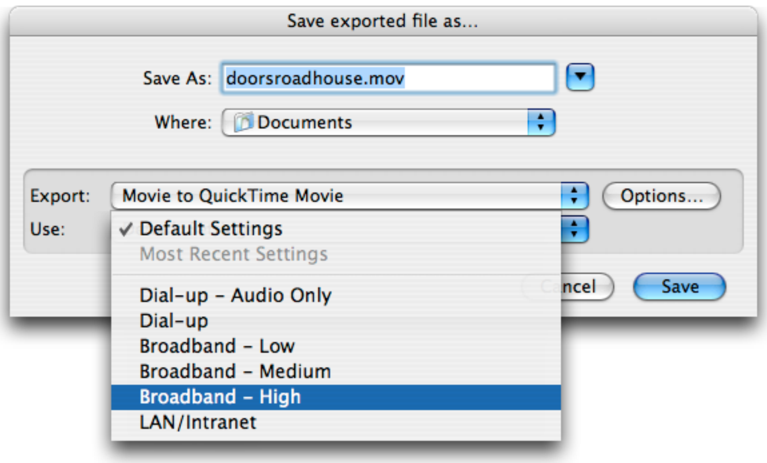
Slideshow
Use the left and right arrows to direct the presentation.

[✓] Display full screen controls

Hide After: 2 seconds

- **Concurrent exports**. Users can export multiple files at once—and continue with their next playback or editing task. Figure 2-7 shows the default export settings for exporting a movie to a QuickTime movie.

Figure 2-7 The default export settings for exporting a movie to a QuickTime movie



Save As: doorsroadhouse.mov

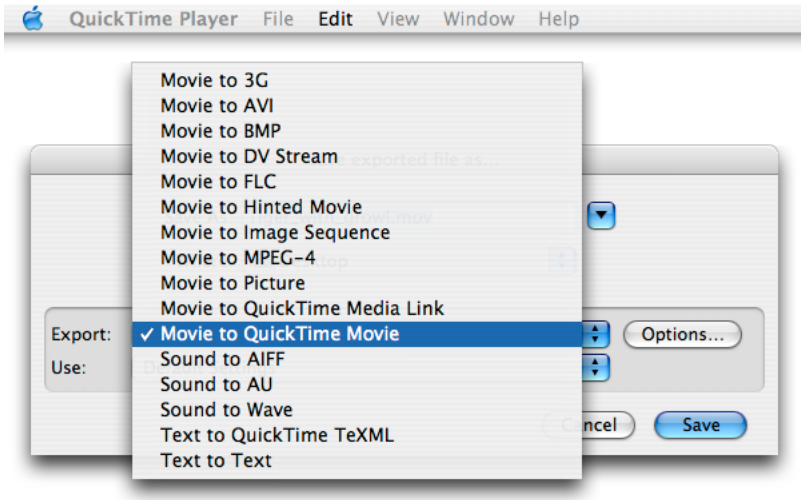
Where: Documents

Export: Movie to QuickTime Movie

Use:
✓ Default Settings
Most Recent Settings
Dial-up – Audio Only
Dial-up
Broadband – Low
Broadband – Medium
Broadband – High
LAN/Intranet

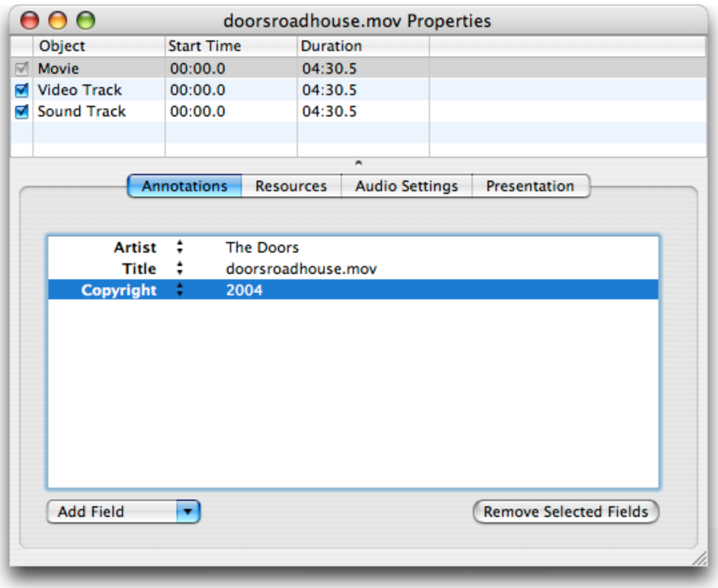
- The export options enable Pro users to export to a variety of image, text, audio, and movie formats, as shown in Figure 2-8.

Figure 2-8 Export options available in QuickTime Pro



■ **Enhanced and redesigned interface for movie settings.** The Movie Properties window has been redesigned to facilitate movie authoring. Figure 2-9 illustrates the Movie Properties dialog, with annotations of the movie selected.

Figure 2-9 The Movie Properties dialog

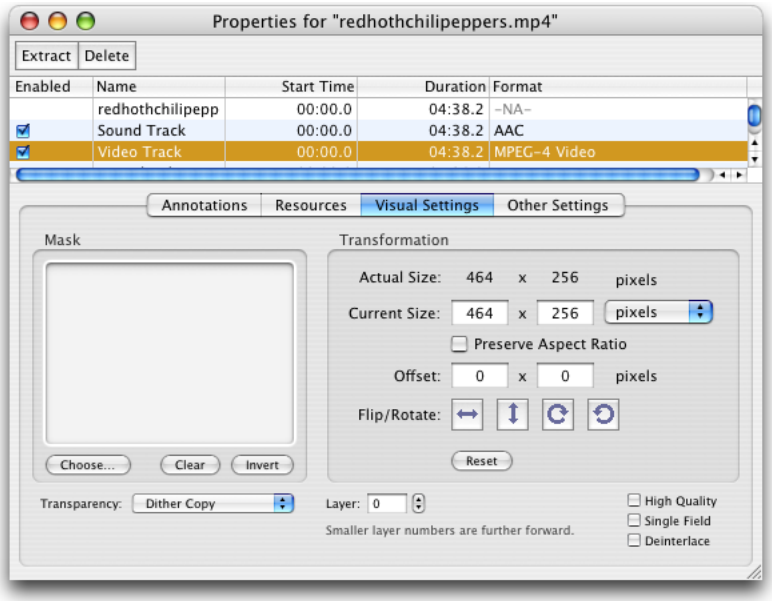


Object	Start Time	Duration
Movie	00:00.0	04:30.5
Video Track	00:00.0	04:30.5
Sound Track	00:00.0	04:30.5

Artist	The Doors
Title	doorsroadhouse.mov
Copyright	2004

■ **New options for image manipulation** in the Visual Settings pane of the Movie Properties dialog of a video track, as shown in Figure 2-10.

Figure 2-10 QuickTime Pro visual settings options for image control and manipulation



Properties for "redhothchilipeppers.mp4"

Enabled	Name	Start Time	Duration	Format
	redhothchilipepp	00:00.0	04:38.2	-NA-
✓	Sound Track	00:00.0	04:38.2	AAC
✓	Video Track	00:00.0	04:38.2	MPEG-4 Video

Transformation

Actual Size: 464 x 256 pixels

Current Size: 464 x 256 pixels

☐ Preserve Aspect Ratio

Offset: 0 x 0 pixels

Flip/Rotate:

Transparency: Dither Copy

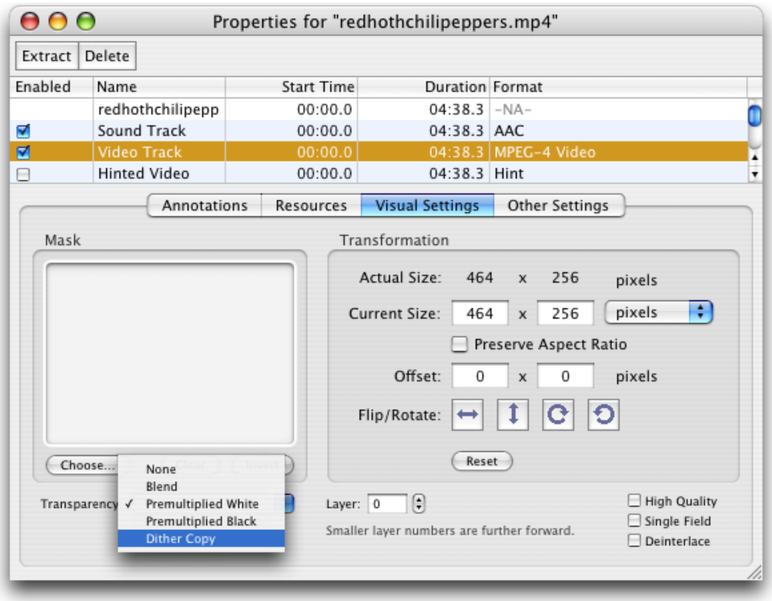
Layer: 0

Smaller layer numbers are further forward.

☐ High Quality
☐ Single Field
☐ Deinterlace

In addition, users are provided with other options to manipulate and control image transparency in QuickTime movies and image files, shown in Figure 2-11.

Figure 2-11 QuickTime Pro user options for controlling image transparency



Properties for "redhothchilipeppers.mp4"

Enabled	Name	Start Time	Duration	Format
	redhothchilipepp	00:00.0	04:38.3	-NA-
✓	Sound Track	00:00.0	04:38.3	AAC
✓	Video Track	00:00.0	04:38.3	MPEG-4 Video
☐	Hinted Video	00:00.0	04:38.3	Hint

Transformation

Actual Size: 464 x 256 pixels

Current Size: 464 x 256 pixels

☐ Preserve Aspect Ratio

Offset: 0 x 0 pixels

Flip/Rotate:

Transparency: ✓ Premultiplied White

Layer: 0

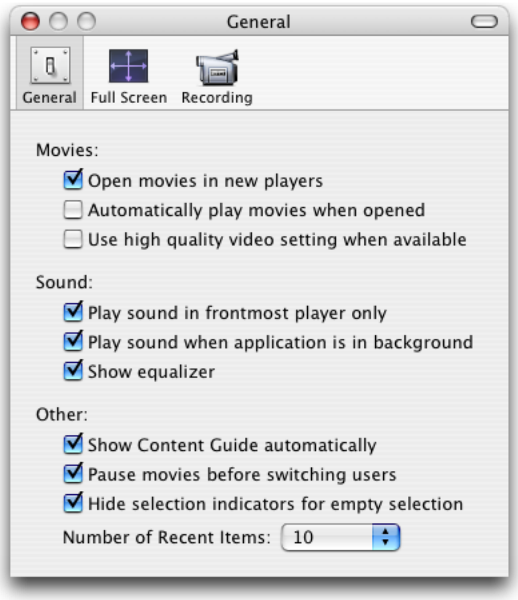
Smaller layer numbers are further forward.

☐ High Quality
☐ Single Field
☐ Deinterlace

Other Changes and Enhancements

QuickTime Preferences now has the option "Use high quality video setting when available." Users can set this as the default for displaying high-quality video tracks, such as DV. Figure 2-12 shows the options available in the General pane of QuickTime Player Preferences.

Figure 2-12 The General pane in QuickTime Player Preferences



Movies:
- [✓] Open movies in new players
- [ ] Automatically play movies when opened
- [ ] Use high quality video setting when available

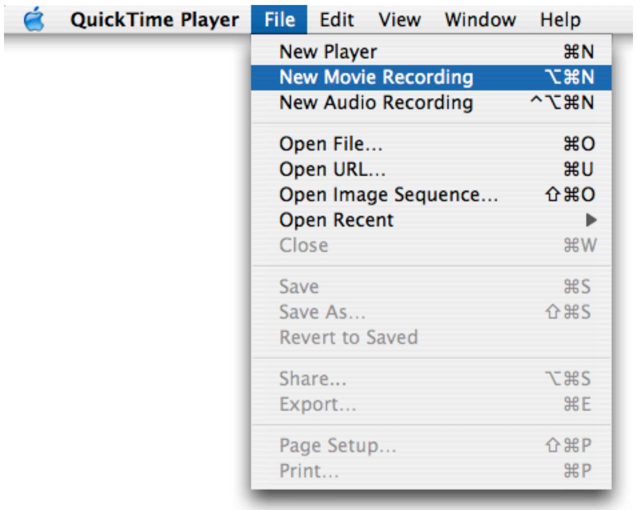
Sound:
- [✓] Play sound in frontmost player only
- [✓] Play sound when application is in background
- [✓] Show equalizer

Other:
- [✓] Show Content Guide automatically
- [✓] Pause movies before switching users
- [✓] Hide selection indicators for empty selection

Number of Recent Items: 10

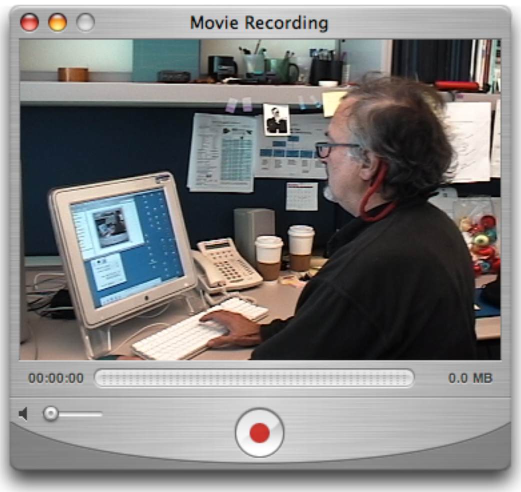
Figure 2-13 shows the new File menu in QuickTime Player. The Open File command enables users to open any of a number of digital media types that QuickTime supports, including movies, still images, VR panoramas, Flash, and so on.

Figure 2-13 The new File menu in the QuickTime Player with New Movie Recording selected



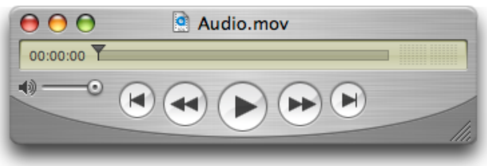
Choosing the File > New Movie Recording menu item enables you to record video from an external digital video camera. Recording is transparent and easy to use, as QuickTime automatically recognizes the device and opens a new QuickTime Player with a red button in the lower center, as shown in Figure 2-14. The Player also displays the current recording duration, as well as the size of the recording in megabytes.

Figure 2-14 Movie recording from a digital device in a new QuickTime player



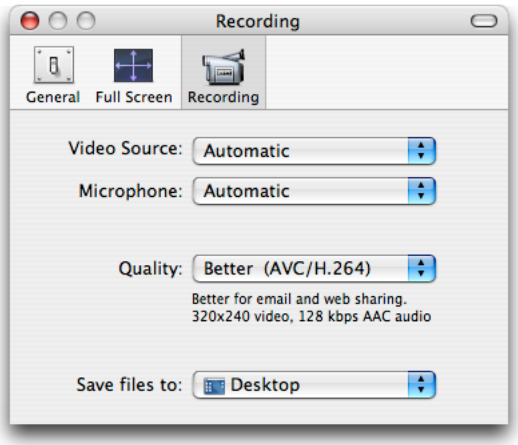
Choosing the File > New Audio Recording menu item enables you to record audio from an external or internal audio device. Once recording begins, a new QuickTime Player appears, as shown in Figure 2-15.

Figure 2-15 Audio recording



To change the video or audio source for your recording, or to specify the quality of recording you want, you choose QuickTime Player > Preferences > Recording, as shown in Figure 2-16.

Figure 2-16 Recording preferences

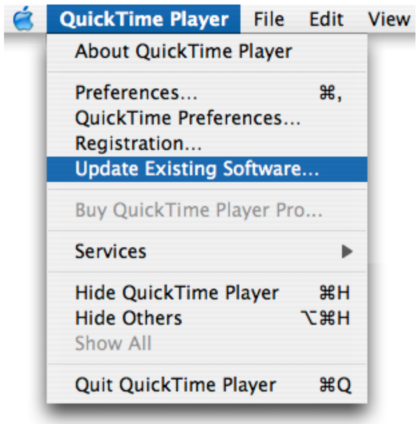


Video Source: Automatic
Microphone: Automatic
Quality: Better (AVC/H.264)
Better for email and web sharing.
320x240 video, 128 kbps AAC audio
Save files to: Desktop

The Save dialog that the Save As command opens now has a "Make movie as a self-contained," selected by default, which is a change from previous versions of QuickTime Player.

Choosing the File > Update Existing Software menu item shown in Figure 2-17 lets you update the version to the latest version of QuickTime available through Software Update.

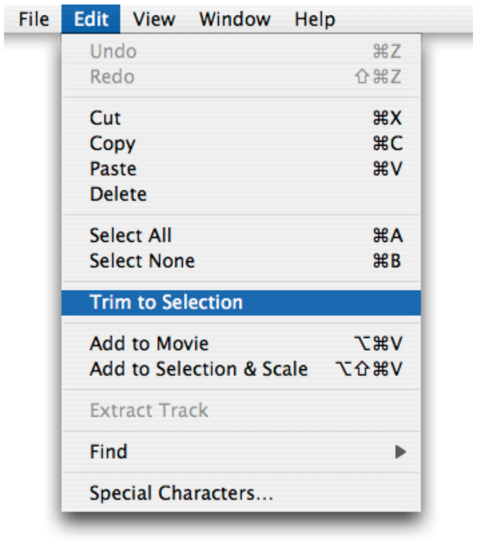
Figure 2-17 The new QuickTime Player menu



The Edit menu has changed from previous versions of QuickTime, as shown in Figure 2-18.

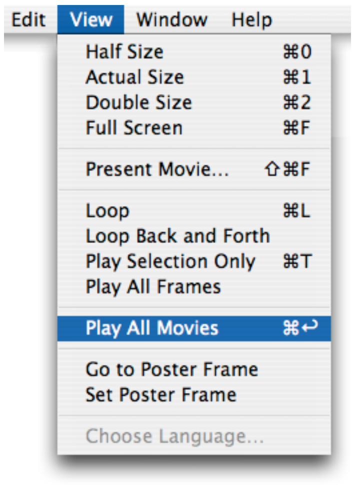
Support for multiple undos is now provided. There is a command Trim to Selection (instead of Trim) and there is no longer a Replace command (users can't do delete and paste as a single operation).

Figure 2-18 The QuickTime Pro Edit menu items with Trim to Selection selected



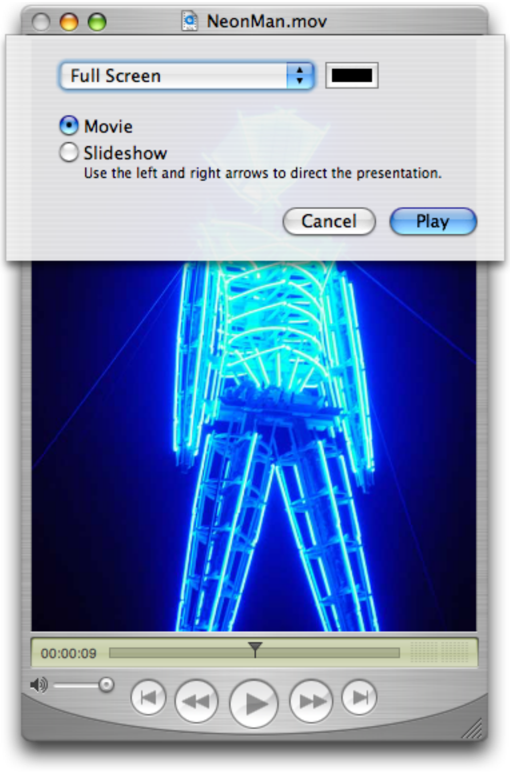
The Movie menu has been renamed and is now the View menu, as shown in Figure 2-19. The Show Movie Properties command has been moved to the Windows menu. Note that this overrides any full screen settings made in user preferences for the current presentation only.

Figure 2-19 The View menu options



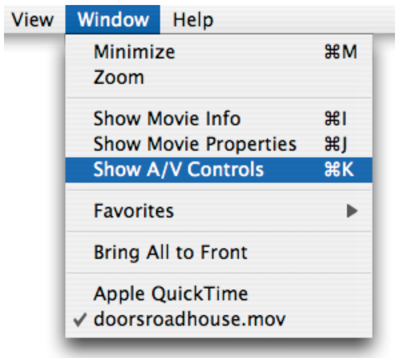
The Present Movie command now opens a sheet as shown in Figure 2-20. The same functionality as in previous versions is provided.

Figure 2-20 The Present Movie sheet



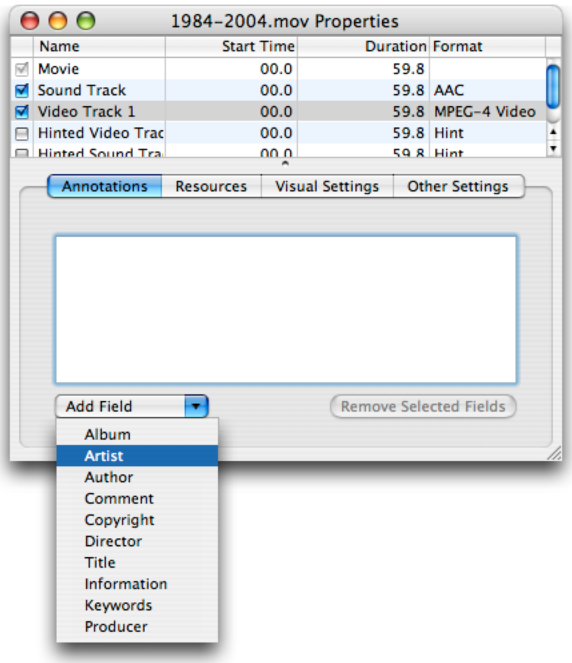
The Window menu (Figure 2-21) now provides commands for getting movie properties and showing audio/video controls.

Figure 2-21 The Window menu with Show A/V Controls selected



Choosing the Window > Show Movie Properties menu item and selecting Video Track 1 (shown in Figure 2-22) enables you to specify certain properties of that track. For example, if you select Annotations and want to add a field, you have multiple choices, including Album, Artist, Author, and so on.

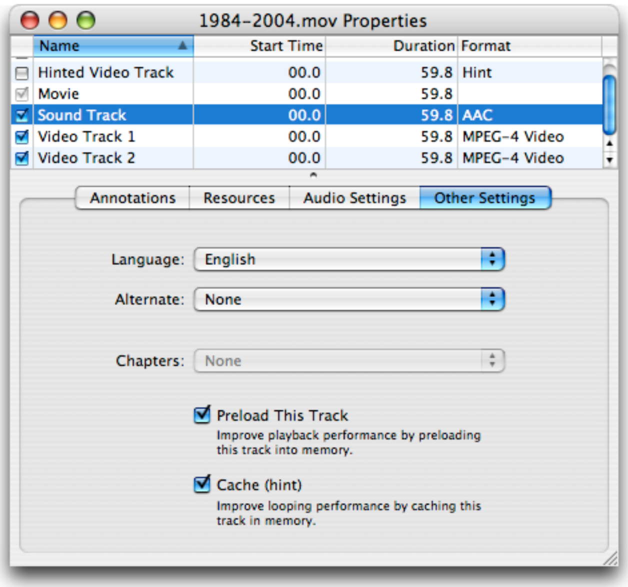
Figure 2-22 Movie properties with annotations and added fields selected


1984-2004.mov Properties

	Name	Start Time	Duration	Format
✓	Movie	00.0	59.8	
✓	Sound Track	00.0	59.8	AAC
✓	Video Track 1	00.0	59.8	MPEG-4 Video
☐	Hinted Video Trac	00.0	59.8	Hint
☐	Hinted Sound Tra	00.0	59.8	Hint

Choosing the Window > Show Movie Properties menu item and selecting Video Track 1 (shown in Figure 2-23) with Other Settings selected enables you to specify certain properties of that track, including language, preloading of the track, caching, and so on.

Figure 2-23 Other settings available in the movie properties pane


1984-2004.mov Properties

	Name	Start Time	Duration	Format
☐	Hinted Video Track	00.0	59.8	Hint
✓	Movie	00.0	59.8	
✓	Sound Track	00.0	59.8	AAC
✓	Video Track 1	00.0	59.8	MPEG-4 Video
✓	Video Track 2	00.0	59.8	MPEG-4 Video

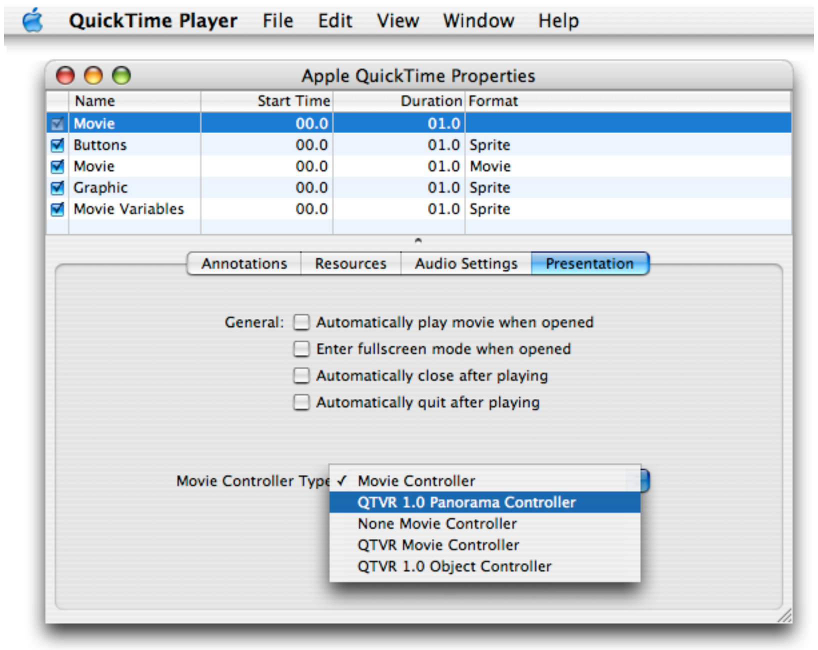
Language: English
Alternate: None
Chapters: None

☑ Preload This Track
Improve playback performance by preloading this track into memory.

☑ Cache (hint)
Improve looping performance by caching this track in memory.

The movie properties window has been reorganized, as shown in Figure 2-24. The Presentation pane provides users with four choices for presenting movie, as well as options for displaying various types of movie controllers.

Figure 2-24 The Presentation pane with a QTVR controller selected



	Name	Start Time	Duration	Format
✓	Movie	00.0	01.0	
✓	Buttons	00.0	01.0	Sprite
✓	Movie	00.0	01.0	Movie
✓	Graphic	00.0	01.0	Sprite
✓	Movie Variables	00.0	01.0	Sprite

General: ☐ Automatically play movie when opened
☐ Enter fullscreen mode when opened
☐ Automatically close after playing
☐ Automatically quit after playing

Movie Controller Type: ✓ Movie Controller

- **New selection handles and fade in/out behavior**. When the user moves the mouse over a selection of the movie, ticks appear that indicate you can make a selection over that area. When you move the mouse over the playbar, the movie will fade the selection indicators in and out. Users can also set in and out points now by placing the current time marker and typing I or O.

New QuickTime Kit Framework

QuickTime 7 introduces Cocoa developers to a new QuickTime Kit framework (`QTKit.framework`). The QuickTime Kit is a Objective-C framework with a rich API for manipulating time-based media.

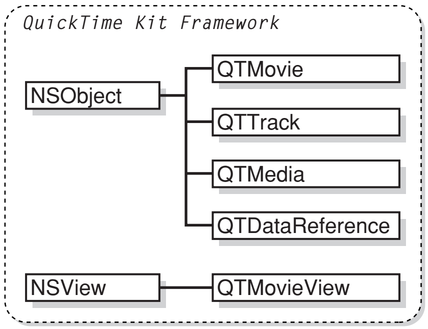
At a basic level, QuickTime Kit provides support for displaying and editing QuickTime movies, relying on abstractions and data types that are already familiar to many Cocoa programmers, such as delegation and notification. QuickTime Kit introduces new data types for QuickTime-related operations only when necessary.

Specifically, two QuickTime Kit classes––`QTMovie` and `QTMovieView`––are intended to replace the existing Application Kit classes `NSMovie` and `NSMovieView`.

The QuickTime Kit framework is new in Mac OS X v10.4 but is also backward-compatible with Mac OS X v10.3 (Panther) as well.

The QuickTime Kit framework provides a set of Objective-C classes and methods designed for the basic manipulation of media, including movie playback, editing, import and export to standard media formats, among other capabilities. The QuickTime Kit framework is at once powerful, yet easy to include in your Cocoa application. Figure 2-25 shows the QuickTime Kit framework's class hierarchy.

Figure 2-25 The QuickTime Kit framework class hierarchy

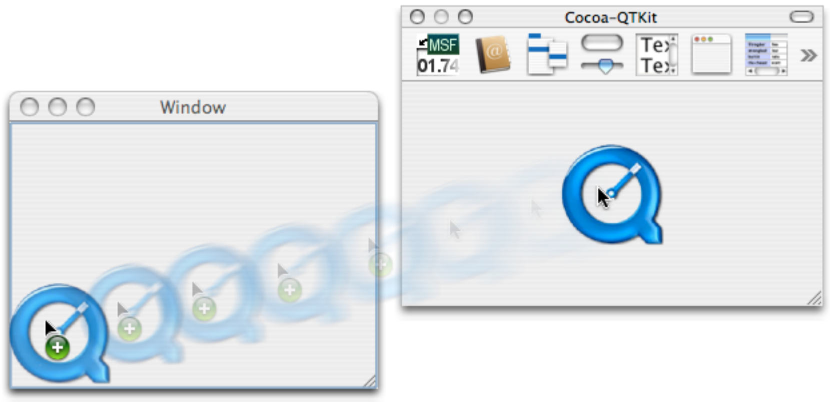


Although the QuickTime Kit framework contains only five classes, you can use these classes and their associated methods, notifications, and protocols to accomplish a broad range of tasks, such as displaying, controlling, and editing QuickTime movies in your Cocoa applications.

A QTKit palette is also provided in Interface Builder that lets you simply drag a QuickTime movie object, complete with a controller for playback, into a window, and then set attributes for the movie––all of this without writing a single line of code.

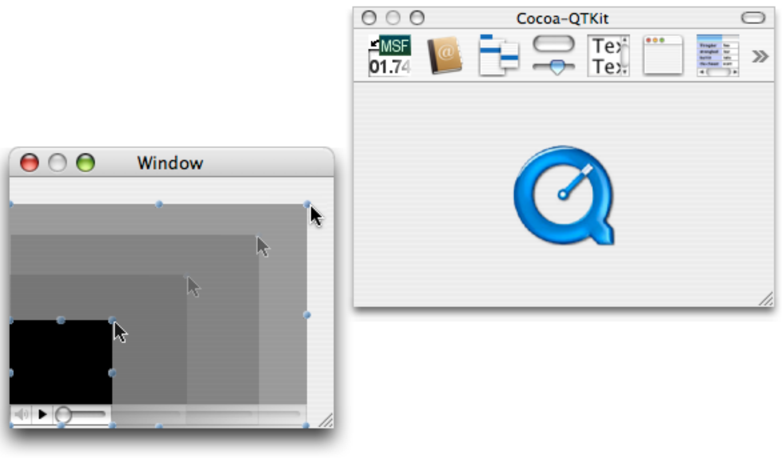
Figure 2-26 shows an animated version of what happens in Interface Builder when you drag the QuickTime object from the QTKit palette to the application window.

Figure 2-26 The QuickTime object dragged to the application window



After you drag the QuickTime object into the application window, you have a QuickTime movie view object with a control bar in the bottom-left corner of the window, as shown in Figure 2-27. By dragging the QuickTime movie view object by its corner handle to the upper-right corner of the window, the entire window fills up so that the movie view object with its control bar is visible.

Figure 2-27 The QuickTime movie view object dragged to fill the entire contents of the window



The QuickTime Kit framework is documented in the QuickTime Kit Reference in conformance with the standards established for Apple's Cocoa documentation suite. A tutorial for using the new framework, QuickTime Kit Programming Guide, is also available online and in PDF format. You can learn how to take advantage of the new QuickTime Kit framework classes and methods and build your own QTKitPlayer application, as well as learn how to extend its functionality.

Audio Enhancements

QuickTime 7 breaks free of the limitations of the Sound Manager, adding many new features and capabilities that developers can take advantage of in their audio playback and capture applications.

Notably, QuickTime 7 now supports **high-resolution audio**, that is, audio sampled at sample rates higher than 64 kHz and up to 192 kHz, with up to 24 channels and support for surround sound. This is in stark contrast to the implementation of the Sound Manager, which only supported mono and stereo. High-resolution audio is supported by Apple's Core Audio technology.

The result of these new audio enhancements is as follows:

- A much richer approach to sound in QuickTime, with support for higher sampling rates, such as 96 kHz and 192 kHz, multiple channels and multiple channel layouts, including 5.1 surround sound and up to 24 discrete channels, meaning channels without any layout imposed on them. Support is also provided for a variety of more accurate audio representations, such as 24-bit uncompressed audio, during capture, playback, and export. Synchronization and access to uncompressed audio on a per-sample basis is also greatly improved, including access to raw PCM audio samples from VBR-compressed audio sources.
- The introduction of a new abstraction layer: the audio context. An audio context represents a connection to a particular audio device. Using an audio context allows you to easily connect a movie to an audio device.
- A more flexible architecture for capturing audio. For instance, multiple sequence grabber audio channels `SGAudioMediaType`) can capture from a single device at the same time, even if the device doesn't permit multiple clients directly, and devices with different channel layouts or different PCM audio formats can be interconnected seamlessly.

- Conversion of audio from one format to another on the fly, performing channel mix-down or remapping, upsampling or downsampling, and sample conversion as needed. This conversion can be performed during export, or as part of the output chain to a device with different playback characteristics than the stored audio, or as part of the capture and storage chain to map input from one or more devices into one or more storage formats.

Most components, with a few exceptions such as streaming and MPEG-4 exporting, will be able to make use of these new capabilities immediately. This release of QuickTime updates a number of components so that it is possible to capture, play back, edit, and export a broad variety of enhanced audio right away.

In brief, QuickTime 7 includes the following enhancements, discussed in this section:

- A new abstraction layer for audio
- A new sound description
- A suite of sound description functions
- New movie property to prevent pitch-shifting
- New functions for gain, balance, and mute
- New level and frequency metering API
- New audio extraction and conversion API
- New audio compression configuration component
- New movie export properties to support high-resolution audio
- New sequence grabber component for audio (`SGAudioMediaType`)

New Abstraction Layer For Audio

QuickTime 7 introduces the **audio context**––a new abstraction that represents playing to an audio device.

As defined, a QuickTime audio context is an abstraction for a connection to an audio device. This allows you to work more easily and efficiently with either single or multiple audio devices in your application.

To create an audio context, you call `QTAudioContextCreateForAudioDevice` and pass in the UID of the device, which is typically a `CFString`. An audio context is then returned. You can then pass that audio content either into `NewMovieFromProperties`, as you would pass in a visual context, or you can open your movie however you would normally open it and call `SetMovieAudioContext`. What that does is route all the sound tracks of the movie to that particular device.

Note that if you want to route two different movies to the same device, you cannot use the same audio context because the audio context is a single connection to that device. What you do is call `QTAudioContextCreateForAudioDevice` again and pass in the same device UID to get another `AudioContext` for the same device, and pass that to your second movie.

High-Resolution Audio Support

High-resolution audio makes use of an enhanced sound description with the ability to describe high sampling rates, multiple channels, and more accurate audio representation and reproduction.

Significantly, the new sound description has larger fields to describe the sampling rate and number of channels, so that the sound description is no longer the limiting factor for these characteristics.

The sound description has built-in support for variable-bit-rate (VBR) audio encoding with variable-duration compressed frames. Extensions to the sound description allow you to describe the spatial layout of the channels, such as quadraphonic and 5.1 surround sound, or to label channels as **discrete**––that is, not tied to a particular geometry. For more information, see "`SoundDescriptionV2`" (page 262).

New movie audio properties include a summary channel layout property, providing a nonredundant listing of all the channel types used in the movie—such as L/R for stereo, or L/R/Ls/Rs/C for 5-channel surround sound—and a device channel layout, listing all the channel types used by the movie's output device.

Figure 2-28 shows the layout of surround speakers. The terminology is defined in Table 1-1.

Figure 2-28 Layout of surround speakers

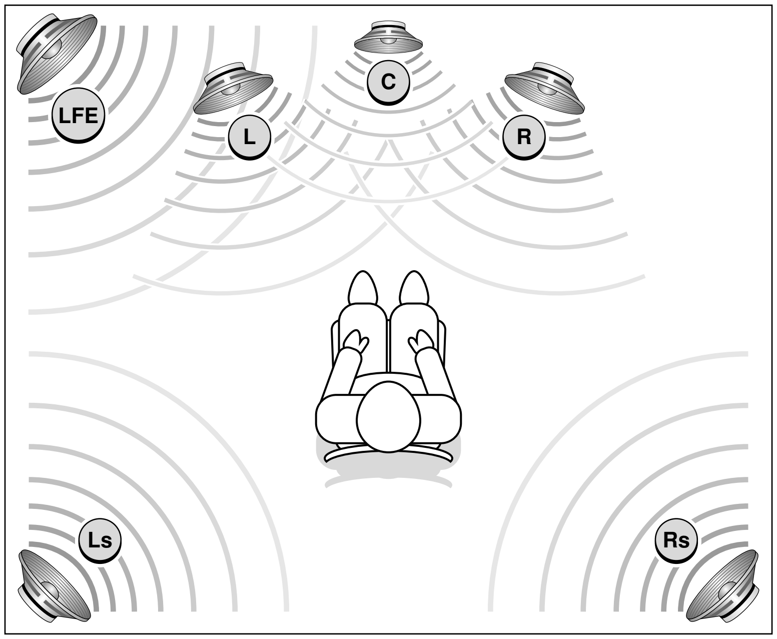


Table 2-1 Surround sound definitions

Speaker	Definition
L	Left speaker
R	Right speaker
C	Center speaker
Ls	Left surround speaker
Rs	Right surround speaker
LFE	Sub-woofer (Note that LFE is an abbreviation for low-frequency effects)

The new sound description is supported by the data types and structures used in the Core Audio framework for Mac OS X (see Core Audio documentation). While the Core Audio API itself is not available to Windows programmers, QuickTime for Windows may include the relevant data structures, such as audio buffers and stream descriptions, audio time stamps and channel layouts, and so on, described in the Core Audio documentation.

A suite of functions has been included to support the handling of sound descriptions opaquely.

Playback

Playback at the high level is automatic and transparent; if you play a movie that contains 96 kHz or 192 kHz sound, it should just work. You should not have to modify your code. The same is true for cut-and-paste editing. If the chosen output device does not support the channel layout, sampling rate, or sample size of the movie audio, mix-down and resampling are performed automatically.

Import of high-resolution audio is automatic, provided the import component has been updated to support high-resolution audio.

Export

Export of high-resolution audio is likewise transparent at the high level. Export at the lower levels requires some additional code. Your application must "opt in" to the new audio features explicitly if it "talks" directly to an export component instance. You do this by calling `QTSetComponentProperty` on the exporter component instance and passing in the `kQTMovieExporterPropertyID_EnableHighResolutionAudioFeatures` property. This is illustrated in the code sample Listing 2-1 (page 37).

Capture

Capturing high-resolution audio requires new code to configure and use the new sequence grabber component for audio. The new audio capture API offers a number of improvements, including the ability to share an input device among multiple sequence grabber channels and the usage of multiple threads for increased efficiency.

When all components in a chain are able to work with high-resolution audio, clock information can be preserved across operations for sample-accurate synchronization.

Sound Description Creation and Accessor Functions

QuickTime 7 provides new functions that let you create, access, and convert sound descriptions.

Sound descriptions can take three basic inputs: an `AudioStreamBasicDescription`, a channel layout, and magic cookie. Sound descriptions are now treated as if they are opaque. In QuickTime 7, when you are handed a sound description, for example, you don't have to go in and look at the version field.

If you want to create a sound description, you can simply hand it an `AudioStreamBasicDescription`, an optional channel layout if you have one, and an optional magic cookie if you need one for the described audio format. Note that it is the format (codec) of the audio that determines whether it needs a magic cookie, not the format of the sound description.

By calling QTSoundDescriptionCreate (page 230), you can make a sound description of any version you choose––for example, one that is of the lowest possible version, given that it is stereo and 16-bit, or one of any particular version you want or request.

The main point about the new API is the capability provided to create a sound description and the usage of new property getters and setters. To accomplish this, follow these steps:

1. Get an AudioStreamBasicDescription from a sound description.
2. Get a channel layout from a sound description (if there is one).
3. Get the magic cookie from magic cookie (if there is one).

At this point, you have all the information you need to talk to Core Audio about this audio. You can also:

1. Get a user-readable textual description of the format described by the SoundDescription.
2. Add or replace a channel layout to an existing sound description. For example, this is what QuickTime Player does in the properties panel where the user can change the channel assignments.
3. Add a magic cookie to a sound description. (This is not needed very often unless you are writing a movie importer, for example.)

To convert an existing QuickTime sound description into the new V2 sound description, you call QTSoundDescriptionConvert (page 230). This lets you convert sound descriptions from one version to another.

For a description of versions 0 and 1 of the SoundDescription record, see the documentation for the QuickTime File Format.

For a description of version 2 of the SoundDescription record, see "SoundDescriptionV2" (page 262). For details of the sound description functions, see QTSoundDescriptionCreate (page 230) and QTSoundDescriptionConvert (page 230).

Audio Playback Enhancements

In addition to playing back high-resolution audio, QuickTime 7 introduces the following audio playback enhancements:

- The ability to play movies at a nonstandard rate without pitch-shifting the audio.
- Getting and setting the gain, balance, and mute values for a movie, or the gain and mute values for a track.
- Providing audio level and frequency metering during playback.

Preventing Pitch-Shifting

A new property is available for use with the NewMovieFromProperties function: kQTAudioPropertyID_RateChangesPreservePitch. When this property is set, changing the movie playback rate will not result in pitch-shifting of the audio. This allows you to fast-forward through a movie without hearing chipmunks.

Setting this property also affects playback of scaled edits, making it possible to change the tempo of a sound segment or scale it to line up with a video segment, for example, without changing the pitch of the sound.

Gain, Mute, and Balance

New functions are available to set the left-right balance for a movie, set the gain for a movie or track, or to mute and unmute a movie or track without changing the gain or balance settings.

The gain and mute functions duplicate existing functions for setting track and movie volume, but the new functions present a simpler and more consistant programmer interface.

For example, to mute the movie using the old `SetMovieVolume` function, you would pass in a negative volume value; to preserve the current volume over a mute and unmute operation, you had to first read the volume, then negate it and set it for muting, then negate it and set it again to unmute. By comparison, the new `SetMovieAudioMute` function simply mutes or unmutes the movie without changing the gain value.

Note: The values set using these functions are not persistent; that is, they are not saved with the movie.

For details, see

- `GetTrackAudioGain` (page 89)
- `SetTrackAudioGain` (page 252)
- `GetTrackAudioMute` (page 89)
- `SetTrackAudioMute` (page 252)
- `GetMovieAudioGain` (page 83)
- `SetMovieAudioGain` (page 246)
- `GetMovieAudioMute` (page 84)
- `SetMovieAudioMute` (page 247)
- `GetMovieAudioBalance` (page 80)
- `SetMovieAudioBalance` (page 244)

Level and Frequency Metering

It is now easy to obtain real-time measurements of the average audio output power level in one or more frequency bands.

You can specify the number of frequency bands to meter. QuickTime divides the possible frequency spectrum (approximately half the audio sampling rate) into that many bands. You can ask QuickTime for the center frequency of each resulting band for display in your user interface.

You can measure the levels either before or after any mix-down or remapping to an output device. For example, if you are playing four-channel surround sound into a stereo output device, you might want to meter the audio levels of all four channels, or you might prefer to see the actual output values delivered to the stereo device.

To use the frequency metering API, follow these steps:

1. Set the number of frequency bands to meter using `SetMovieAudioFrequencyMeteringNumBands`.
2. Call `GetMovieAudioFrequencyMeteringBandFrequencies` if you need to know the frequencies of the resulting bands.
3. Finally, make periodic calls to `GetMovieAudioFrequencyLevels` to obtain measurements in all specified bands. You can obtain either the average values, the peak hold values, or both.

For details, see

- `GetMovieAudioVolumeMeteringEnabled` (page 85)
- `SetMovieAudioVolumeMeteringEnabled` (page 248)
- `GetMovieAudioVolumeLevels` (page 84)
- `GetMovieAudioFrequencyMeteringNumBands` (page 82)
- `SetMovieAudioFrequencyMeteringNumBands` (page 246)
- `GetMovieAudioFrequencyMeteringBandFrequencies` (page 82)
- `GetMovieAudioFrequencyLevels` (page 81)

Audio Conversion, Export, and Extraction

The new audio extraction API lets you retrieve mixed, uncompressed audio from a movie.

Note that the audio extraction API currently *only* mixes audio from sound tracks. Other media types, such as muxed MPEG-1 audio inside a program stream, are not currently supported.

To use the audio extraction API, follow these steps:

1. Begin by calling `MovieAudioExtractionBegin` (page 185). This returns an opaque session object that you pass to subsequent extraction routines.
2. You can then get the `AudioStreamBasicDescription` for the audio or layout. Note that some properties are of variable size, such as the channel layout, depending on the audio format, so getting the information involves a two-step process.
   a. First, you call `MovieAudioExtractionGetPropertyInfo` (page 189) to find out how much space to allocate.
   b. Next, call `MovieAudioExtractionGetProperty` (page 187) to obtain the actual value of the property.
3. You can use the `AudioStreamBasicDescription` to specify a different uncompressed format than Float 32. This causes the extraction API to automatically convert from the stored audio format into your specified format.
4. Use the `MovieAudioExtractionSetProperty` (page 189) function to specify channel remapping––that is, a different layout––sample rate conversion, and preferred sample size. You can also use this function to specify interleaved samples (default is non-interleaved) or to set the movie time to an arbitrary point.

Note that there are basically two things you set here: an audio stream basic description (ASBD) and a channel layout. (ASBD sets the format, sample, number of channels, interleavings, and so on.)

Setup is now complete. You can now make a series of calls to `MovieAudioExtractionFillBuffer` to receive uncompressed PCM audio in your chosen format.

1. The default is for the first call to begin extracting audio at the start of the movie, and for subsequent calls to begin where the last call left off, but you can set the extraction point anywhere in the movie timeline by calling `MovieAudioExtractionSetProperty` and setting the movie time.
2. `MovieAudioExtractionFillBuffer` will set `kMovieAudioExtractionComplete` in *`outFlags`* when you reach the end of the movie audio.
3. You must call `MovieAudioExtractionEnd` when you are done. This deallocates internal buffers and data structures that would otherwise continue to use memory and resources.

A caveat: Ideally, the uncompressed samples would be bitwise identical whether you obtained the samples by starting at the beginning of the movie and iterating through it, or by randomly setting the movie time and extracting audio samples. This is typically the case, but for some compression schemes the output of the decompressor depends not only on the compressed sample, but the seed value in the decompressor that remains after previous operations.

The current release of QuickTime does not perform the necessary work to determine what the seed value would be when the movie time is changed prior to extracting audio; while the extracted audio is generally indistinguishable by ear, it may not always be bitwise identical.

For details about audio conversion, export, and extraction, refer to the information about the following functions:

- `MovieAudioExtractionBegin` (page 185)
- `MovieAudioExtractionGetPropertyInfo` (page 189)
- `MovieAudioExtractionGetProperty` (page 187)
- `MovieAudioExtractionSetProperty` (page 189)
- `MovieAudioExtractionFillBuffer` (page 186)
- `MovieAudioExtractionEnd` (page 186)

Standard Audio Compression Enhancements

QuickTime 7 introduces a new standard compressor component, `StandardCompressionSubTypeAudio`, that adds the ability to configure high-resolution audio output formats. It uses Core Audio internally instead of the Sound Manager, and has a full set of component properties to make configuration easier, especially when the developer wishes to bring up an application-specific dialog, or no dialog, rather than the typical compression dialog.

This component essentially replaces the `StandardCompressionSubTypeSound` component, which is limited to 1 or 2 channel sound with sampling rates of 65 kHz or less. That component is retained for backward compatability with existing code, but its use is no longer recommended.

The `StandardCompressionSubTypeAudio` component is configured by getting and setting component properties, instead of using GetInfo and SetInfo calls. These properties have a class and ID, instead of just a single selector.

The component property API allows configuration at any level of detail without requiring a user interface dialog or direct communication with low-level components.

For details, refer to the sections "SGAudio Component Property Classes" (page 273)and "SGAudio Component Property IDs" (page 273).

Note: You can also configure the new standard audio compression component by calling `SCSetSettingsFromAtomContainer`. You can pass the new standard audio compression component either a new atom container obtained from `SCGetSettingsAsAtomContainer` or an old atom container returned by calling the same function (`SCGetSettingsAsAtomContainer)` on the old `SubTypeSound` component.

If you use `MovieExportToDataRefFromProcedures`, your getProperty proc will need to support some of these property IDs as new selectors. Note that the Movie Exporter getProperty proc API is not changing to add a class (the class is implied).

Note: Not all properties can be implemented by getProperty procs; the properties that getProperty procs can implement are marked with the word "DataProc". See the inline documentation in `QuickTimeComponents.h` for more information.

Audio Export Enhancements

Some movie export components now support high-resolution audio.

Export of high-resolution audio is transparent at the high level. If you export from a movie containing high-resolution audio to a format whose export component supports it, the transfer of data is automatic; if the export component does not support high-resolution audio, mix-down, resampling, and sound description conversion are automatic.

Export at the lower levels requires some additional code. Your application must "opt in" to the new audio features explicitly if it talks directly to an export component instance. (This is to prevent applications that have inadvisedly chosen to "walk" the opaque atom settings structure from crashing when they encounter the new and radically different structure.) The following code snippet (Listing 2-1) illustrates the opt-in process.

Listing 2-1 Opting in for high-resolution audio export

```
ComponentInstance exporterCI;
ComponentDescription search = { 'spit', 'MooV', 'appl', 0, 0 };
Boolean useHighResolutionAudio = true, canceled;
OSStatus err = noErr;

Component c = FindNextComponent(NULL, &search);
exporterCI = OpenComponent(c);

// Hey exporter, I understand high-resolution audio!!
(void) QTSetComponentProperty(// disregard error
        exporterCI,
        kQTPropertyClass_MovieExporter,
        kQTMovieExporterPropertyID_EnableHighResolutionAudioFeatures,
        sizeof(Boolean),
        &useHighResolutionAudio);

err = MovieExportDoUserDialog(exporterCI, myMovie, NULL, 0, 0, &canceled);
```

For additional details, see "`Movie Exporter Properties`" (page 272).

Audio Capture Enhancements

There is a new sequence grabber channel component (`'sgch'`) subtype for audio, `SGAudioMediaType` (`'audi'`), which allows capture of high-resolution audio, supporting multi-channel, high sample rate, high accuracy sound. This is intended to replace the older `SoundMediaType` component.

Important: The new component still captures a sound track of type `SoundMediaType` (`'soun'`); only the *sequence grabber* media type changes, not the final track media type.

The new audio channel component has a number of noteworthy features, including:

- audio capture to VBR compressed formats
- enabling or disabling of source channels on a multi-channel input device
- mix-down and remapping of multi-channel audio source material
- discrete and spatial labeling of channels (for example, 5.1 or discrete)
- audio format and sample rate conversion during capture
- sharing of audio input devices among multiple sequence grabber audio channels
- sharing of audio playback devices among multiple sequence grabber audio channels
- notification of audio device hotplug/unplug events
- audio preview of source data or compressed data
- splitting audio channels from a record device to separate tracks in a movie
- redundant capture of multichannel audio to separate tracks in a movie (with independent data rates and compression settings)
- client callbacks of audio pre- and post-mixdown, and pre- and post-conversion with propagation of audio time stamps and audio samples to interested clients
- improved A/V sync
- improved threading model compared with the legacy `SoundMediaType`
- lower latency audio grabs
- reduced dependency on frequent `SGIdle` calls

This new, advanced functionality makes extensive use of Core Audio methodology and data structures.

Configuring Audio Channel Components

The audio channel component can be configured using component properties. This has several advantages over using a sequence grabber panel. For one thing, it can be configured without a user dialog, or using an application-specific dialog. For another, it is possible to test for properties and get or set them dynamically, allowing the same code to configure multiple audio input devices, including unfamiliar devices.

The application does not need to bypass the channel component and connect directly to an input device, such as a `SoundInputDriver`, to set low-level properties. This allows multiple capture channels to share a single input device, and keeps application code from becoming tied to a particular device type.

For a full list of the `SGAudioMediaType` component properties, see "`SGAudio Component Property IDs`" (page 273). For a full list of component property classes, see "`SGAudio Component Property Classes`" (page 273).

Once the component is configured, the audio capture—plus any desired mixdown, format or sample-rate conversion, and compression—take place in a combination of real-time and high-priority threads. Multichannel data is interleaved and samples are put into a queue. You can set up callbacks to watch the data at any of several points in the chain: pre-mixdown, post-mixdown, pre-conversion, or post-conversion.

The actual writing of the captured audio to a storage medium, such as a disk file, takes place during calls to `SGIdle`.

One input device can be shared by multiple sequence grabber channels, as illustrated in Figure 2-29. Because independent mix and conversion stages exist for each sequence grabber audio channel, the sequence grabber audio channels can capture different channel mixes, sampling rates, sample sizes, or compression schemes from the same source. Similarly, multiple sequence grabber audio channels can share a common output device for previewing.

Channel mixdown or remapping, sample conversion, and any compression are all performed on high-priority threads. Each sequence grabber channel receives data from only those audio channels it has requested, in the format it has specified. The following processing may occur in the background:

- software gain adjustment
- mixing
- sample rate conversion
- bit-depth widening or shortening
- float to integer conversion
- byte-order conversion (big-endian to little-endian or vice-versa)
- encoding of frames into compressed packets of data in the specified format
- interleaving

The resulting frames or packets are held in a queue, to be written to file or broadcast stream on the main thread. This is accomplished during calls to `SGIdle`, at which time the audio is chunked and interleaved with any video data being captured.

Sequence Grabber Audio Channel Mapping

Figure 2-29 is a high-level diagram that shows some of the internal workings of the sequence grabber audio channel, such as the Core Audio matrix mixer and the audio converter that lets you convert, compress, and interleave audio, and then queue the audio. From the queue, the audio can be written to disk in desired chunk sizes. One distinct advantage of this process is that you can take a single device and share it among multiple channels. This results in simultaneous recording from multiple devices into multiple tracks in a QuickTime movie. In addition, you can record multiple tracks from a single device.

Figure 2-29 QuickTime audio device sharing among sequence grabber channels

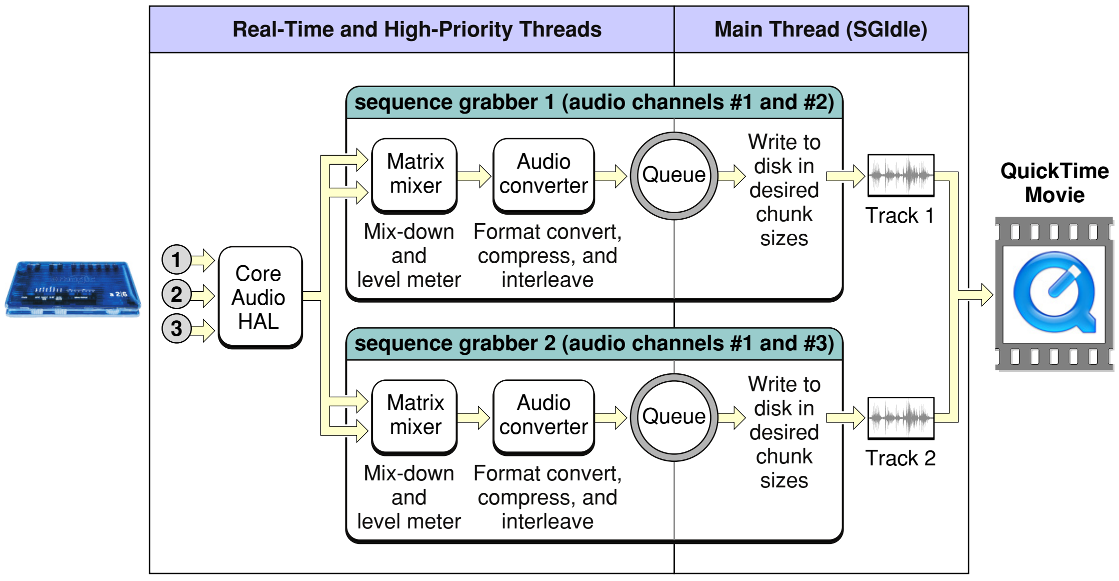

Real-Time and High-Priority Threads
Main Thread (SGIdle)
sequence grabber 1 (audio channels #1 and #2)
Matrix mixer
Audio converter
Queue
Write to disk in desired chunk sizes
Track 1
Mix-down and level meter
Format convert, compress, and interleave
QuickTime Movie
1
2
3
Core Audio HAL
sequence grabber 2 (audio channels #1 and #3)
Matrix mixer
Audio converter
Queue
Write to disk in desired chunk sizes
Track 2
Mix-down and level meter
Format convert, compress, and interleave

Figure 2-30 illustrates a usage case that involves client channel mapping. This shows how a client can instantiate multiple sequence grabber audio channels that share a recording device. This enables the "splitting" of device channels across multiple tracks in a QuickTime movie. In Figure 2-30, there is single recording device, with four channels. The first two channels record into Track 1 in a QuickTime movie. The second sequence grabber audio channel, which records into Track 2 in a QuickTime movie, only wants channel 4 from the recording device, so that you can get one stereo track and one mono track.

In this example, device Track 0 will get into Movie Track 1, while Movie Track 3 has only one slot to fill. You can mix and match different channel map valences in such a way as to disable certain tracks in a movie and get submixes, for example. In code, it looks like this:

```
SInt32 map 1 [ ] = { 0, 1 };
SInt32 map 2 [ ] = { 3 };
```

Figure 2-30 Client channel mapping with "splitting" of device channels

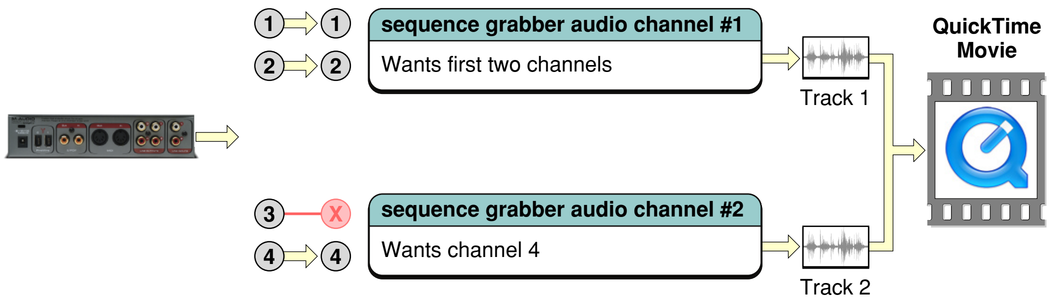


Figure 2-31 shows another usage case that also involves client channel mapping.

A sequence grabber audio channel shown in the illustration can get four channels from a device in any order that makes sense for the client. Consider, for instance, a device that supports four-channels of audio. Using the channel map property IDs ("SGAudio Component Property IDs" (page 273)), you can reorder channels from a recording device to a desired movie channel valence. In code, it looks like this:

```
SInt32 map [ 4 ] = { 3, 2, 1, 0 };
```

Figure 2-31 Channel mapping with reordering of channels

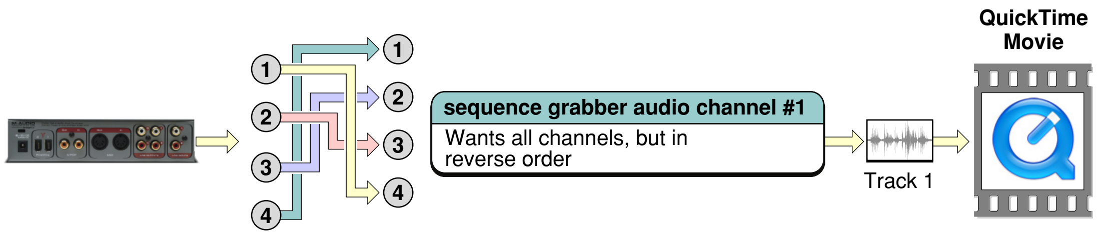

1
2
3
4
1
2
3
4
sequence grabber audio channel #1
Wants all channels, but in reverse order
Track 1
QuickTime Movie

Figure 2-32 shows another example of what you can do with the feature of channel mapping, in this case **mult'-ing**, that is, duplicating channels from a recording device into multiple output channels in QuickTime movie tracks. For instance, you can take advantage of this channel mapping feature if you have one recording device and two sequence grabber audio channels, and they're both going to make the same movie. The first sequence grabber audio channel wants the first stereo pair twice (1, 2, 1, 2), while the second wants the second stereo pair twice (3, 4, 3, 4). In code, it looks like this (zero-based indexing):

```
SInt32 map 1 [ ] = { 0, 1, 0, 1 };
SInt32 map 2 [ ] = { 2, 3, 2, 3 };
```

Figure 2-32 Channel mapping enabling mult-ing

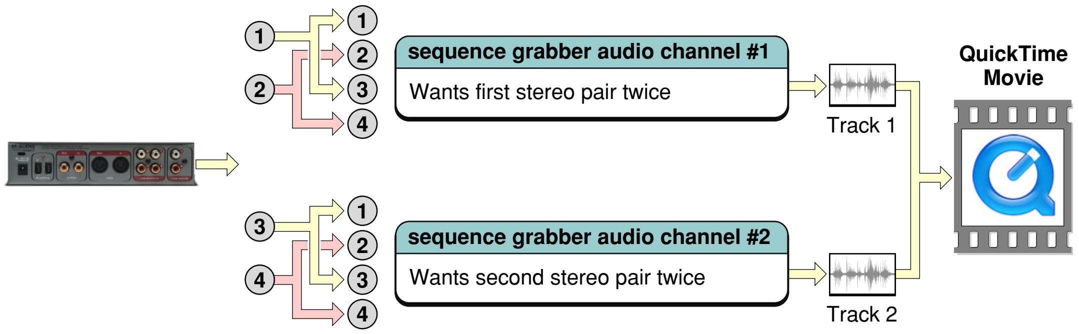

1
2
3
4
sequence grabber audio channel #1
Wants first stereo pair twice
sequence grabber audio channel #2
Wants second stereo pair twice
Track 1
Track 2
QuickTime Movie

Figure 2-33 illustrates the what you can do with multiple mixes. Because you can duplicate device channels onto multiple output tracks, you can create a movie containing multiple mixes of the same source material.

This is useful for a recording situation where you have a six channel recording device and are presenting 5.1 material. You could make a QuickTime movie that has four tracks in it. In this case, the first track is getting the raw, unmixed source––that is, channels one through six. You will have a six discrete channel track, meaning that the first channel plays out to the first speaker, the second channel out to the second speaker, and so on.

In sequence grabber audio channel #2, you'll get a 5.1 mix and apply spatial orientation to the six channels, specifying the speakers to which the audio will play. All four tracks are going into a QuickTime movie. Sequence grabber audio channel #3 presents a stereo mix-down, while sequence grabber audio channel #4 presents a mono mix-down.

Figure 2-33 Channel mapping with simultaneous multiple mixes

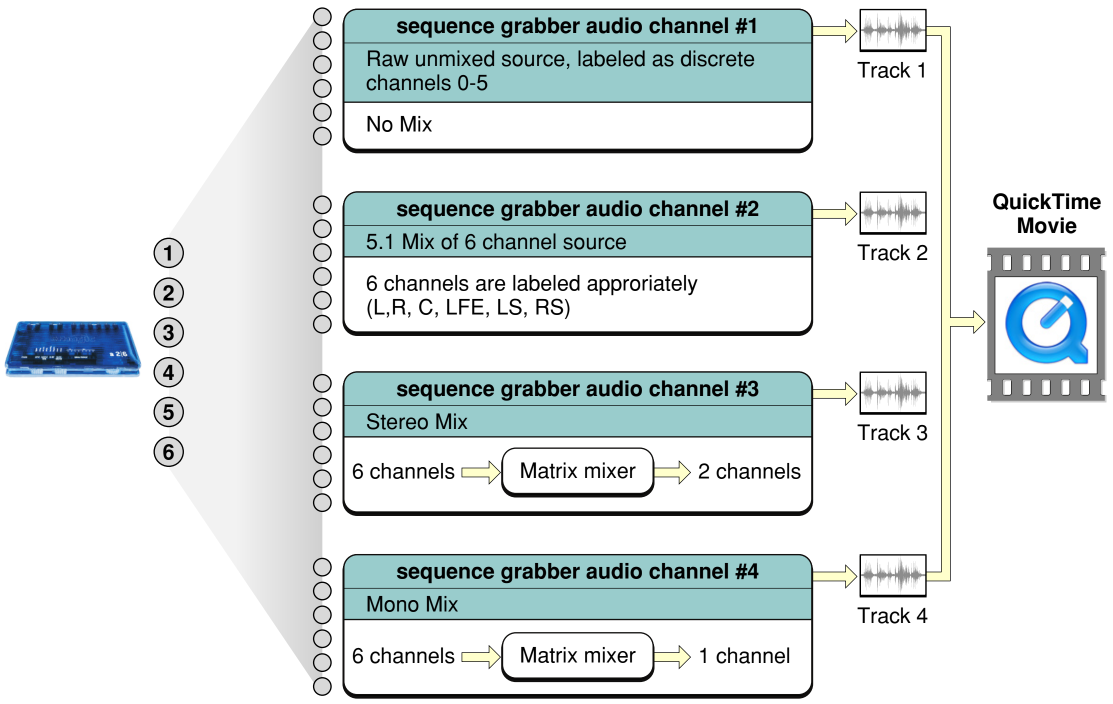

1
2
3
4
5
6
sequence grabber audio channel #1
Raw unmixed source, labeled as discrete channels 0-5
No Mix
sequence grabber audio channel #2
5.1 Mix of 6 channel source
6 channels are labeled appproriately (L,R, C, LFE, LS, RS)
sequence grabber audio channel #3
Stereo Mix
6 channels
Matrix mixer
2 channels
sequence grabber audio channel #4
Mono Mix
6 channels
Matrix mixer
1 channel
Track 1
Track 2
Track 3
Track 4
QuickTime Movie

Figure 2-34 shows channel mapping with multi-date rates, similar to multiple mixes, except that you can also apply compression to the mixes. As a result, you can broadcast multiple streams at once.

Figure 2-34 Channel mapping with multi-data rates

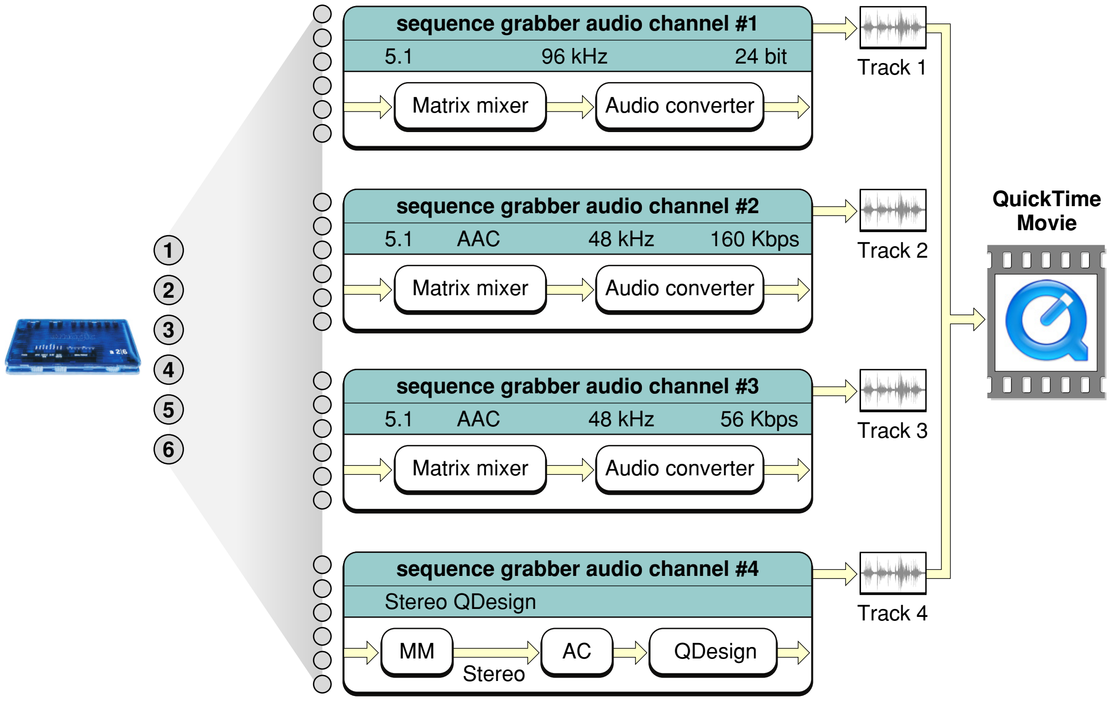


Figure 2-35 shows sequence grabber audio callbacks, which are analogous to the `VideoMediaType` sequence grabber channel video bottlenecks. The callbacks provide developers with different places in the audio chain where they can "pipe in" and look at the samples.

Figure 2-35 Sequence grabber audio callbacks, analogous to sequence grabber video bottlenecks callbacks

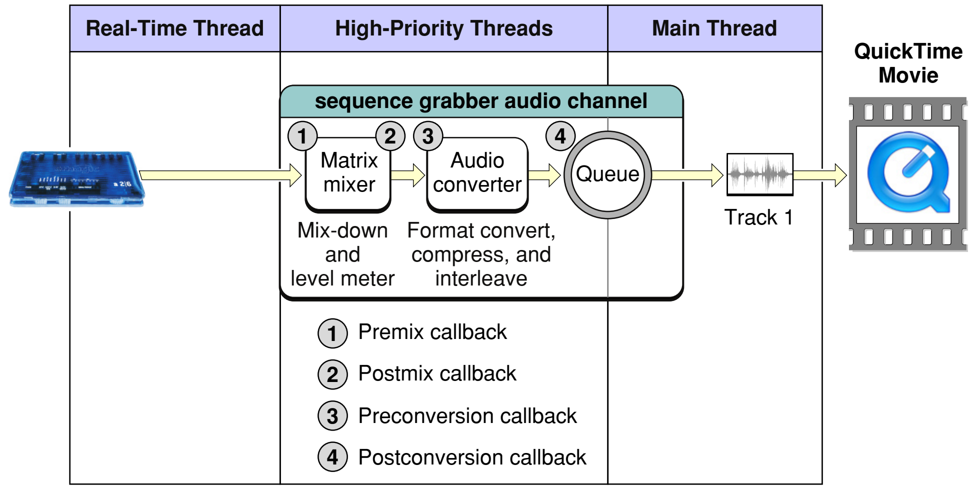
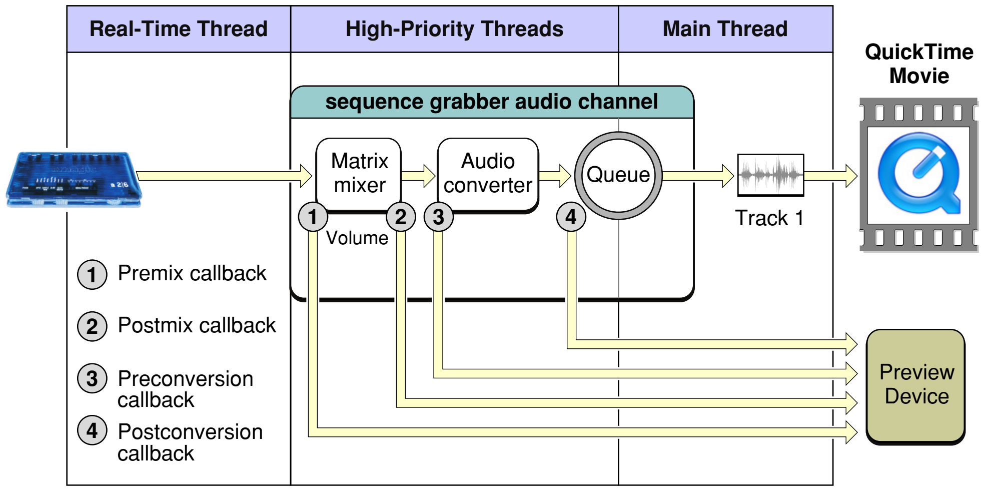


Figure 2-36 shows sequence grabber audio callbacks, with real-time preview. Clients can specify what they want to preview, using the sequence grabber channel play flags.

Figure 2-36 SGAudio callbacks with real-time preview



Using Sequence Grabber Audio Features

To make use of the new sequence grabber audio features, follow these steps:

1. Instantiate a sequence grabber channel of subtype `SGAudioMediaType` (`'audi'`), by calling `SGNewChannel(sg, SGAudioMediaType, &audiChannel)`.

2. Use the QuickTime component property API to obtain a list of available input and preview devices from the sequence grabber channel, by getting the property `kQTSGPropertyID_DeviceListWithAttributes` (`'#dva'`).

3. Use the same component property API to get the input device characteristics and set the desired audio format and device settings. See "`SGAudio Component Property Classes`" (page 273) and "`SGAudio Component Property IDs`" (page 273) for details. Note that this is sometimes a two-stage process, as next described.

4. a. Use `QTGetComponentPropertyInfo` to determine the size of the property value.

   b. Allocate the necessary container and use `QTGetComponentProperty` to obtain the actual value. This is necessary with properties such as channel layout, which is a variable length structure.

5. Call `SGStartRecord` or `SGStartPreview`, enabling the sequence grabber, and then make periodic calls to `SGIdle`.

If you are capturing only sequence grabber audio media, it is no longer necessary to make extremely frequent calls to `SGIdle`, since this function is only used to write the samples to storage, not to capture data from the input device. When capturing video or using an old-style sequence grabber sound media component, however, you must still call `SGIdle` frequently (at a frequency greater than the video sample rate or the sound chunk rate).

By setting the appropriate sequence grabber channel properties and setting up a callback, you can examine samples at various points in the input chain, such as premix, postmix, preconversion, and postconversion. For details, see `SGAudioCallbackProc` (page 256), `SGAudioCallbackStruct` (page 261), "`SGAudio Component Property Classes`" (page 273) and "`SGAudio Component Property IDs`" (page 273).

Video Enhancements

QuickTime 7 introduces a number of important video enhancements, discussed in this section. These include

- Support for compressed video using frame reordering. Support is added for compression, playback, streaming, and low-level access to stored samples.
- A new visual context that provides an abstraction layer that is intended to decouple QuickTime from graphics worlds (GWorlds). This decoupling allows programmers to work in QuickTime without needing to understand QuickDraw, and to more easily render QuickTime directly using engines such as OpenGL.
- Support for H.264 video compression, including QuickTime components for export, playback, and live streaming.

Frame Reordering Video

QuickTime 7 adds support for **frame reordering** video compression. This is a major advance that involves new sample tables for video to allow video frames to have independent decode and display times.

The result of using frame reordering for video compression is improved display, editing, and capture in H.264 and other advanced video codec formats. Enhancements include a new API for working with media sample times, adding and finding samples, and a new Image Compression Manager (ICM) API.

Understanding Frame Reordering Video Compression

QuickTime supports many types of video compression, including spatial compression algorithms, such as photo-JPEG, and temporal compression algorithms, in which some video frames are described completely, while other frames are described in terms of their differences from other video frames.

Up until the introduction of H.264 in QuickTime 7, video frames could be of three kinds:

- I-frames (independently decodable)
- P-frames (predicted from a previous I- or P-frame)
- B-frames (predicted from one past and one future I- or P-frame)

Note: B-frame, I-frame, and P-frame are all video compression methods used by the MPEG standard. **B-frame** is an abbreviation for *bi-directional* frame, or bi-directional predictive frame. B-frames rely on the frames preceding and following them and only contain data that has changed from the preceding frame or is different from data in the next frame.

P-frame is an abbreviation for *predictive* frame, or predicted frame. P-frames follow I-frames and contain only the data that has changed from the preceding I-frame. P-frames rely on I-frames to fill in most of its data.

I-frame, also known as keyframes, is an abbreviation for *intraframe*. An I-frame stores all the data required to display the frame. In common usage, I-frames are interspersed with P-frames and B-frames in a compressed video.

Because B-frames predict from a future frame, that frame has to be decoded before the B-frame, yet displayed after it; this is why frame reordering is needed.The decoded order is no longer the same as the displayed order.The QuickTime support for frame reordering is quite general.In the H.264 codec, the concepts of the direction of prediction, and the numbers of referenced frames, and the kind of frame that is referenced, are all decoupled. In H.264, an encoder may choose to make a stream in which P-frames refer to a future frame, or a B-frame which refers to two past or future frames, for example.

Important: Prior to this release, QuickTime supported self-contained video frames (keyframes, also called sync-frames or I-frames) and frames that depended on previous frames (P-frames). Many modern compressors also make use of frame reordering, in which frames can depend on future frames.Those future frames have to be decoded before the frame in question, but displayed after it––hence the reordering. Traditional B-frames are one example: they depend on a past and a future I- or P-frame.That future I- or P-frame has to be given to the decoder before the B-frame, but is displayed after the B-frame itself. This means that the frames are stored or streamed in decode order, rather than in display order.

For decompressors that don’t use frame reorderings, the decode order and the display order are the same, and QuickTime sample tables are traditionally organized to reflect this. Samples are stored in decode order, which is presumed to be the display order, and the sample tables specify the duration of each sample’s display; the display time is the time when the track begins plus the duration of all previous samples.

The addition of frame reordering support means that QuickTime now has an optional sample table for video that specifies the offset between the decode time and the display time. This allows frames to be stored in decode order but displayed in a different order. The decode time is still the beginning of the track plus the decode duration of all previous samples, but it is now necessary to examine the offset table to determine which samples precede others and calculate the correct display time.

For high-level programmers, this all happens transparently. Developers who work directly with sample numbers and sample times, however, must be aware of this new feature. A new, expanded API is available to support this.

Finding and Adding Samples

Developers who need to work with specific samples based on the samples’ display times, or who are adding samples to a media directly, need to use a different API when working with media that uses frame reorderings.

For example, programmers who use the function `MediaTimeToSampleNum` must instead use the two functions `MediaDecodeTimeToSampleNum` and `MediaDisplayTimeToSampleNum` when working with frame reordering compressed video, as each sample now has a decode time and a display time instead of a single media time (combined decode/display time).

Similarly, when adding samples to a media that permits display offsets, it is necessary to use the new `AddMediaSample2` instead of `AddMediaSample`, as the new function permits the user to pass a display offset and specify properties that are unique to media with display offsets, such as whether subsequent samples are allowed to have earlier display times than the current sample.

Calling one of the old functions that use a single media time value on new-format media that contains display offsets will return the error code `kQTErrMediaHasDisplayOffsets`.

The new API elements all use 64-bit time values, whereas the older API elements use 32-bit values. Calling one of the old functions with a 64-bit time value returns the error code `kQTErrTimeValueTooBig`.

When creating a media for frame reordering compressed video track, pass in the new flag `kCharacteristicSupportsDisplayOffsets`.

For details, see:

- `AddMediaSample2` (page 63)
- `ExtendMediaDecodeDurationToDisplayEndTime` (page 69)
- `GetMediaAdvanceDecodeTime` (page 71)
- `GetMediaDataSizeTime64` (page 72)
- `GetMediaDecodeDuration` (page 73)
- `GetMediaDisplayDuration` (page 73)
- `GetMediaDisplayEndTime` (page 74)
- `GetMediaDisplayStartTime` (page 74)
- `GetMediaNextInterestingDecodeTime` (page 75)
- `GetMediaNextInterestingDisplayTime` (page 76)
- `GetMediaSample2` (page 77)
- `MediaContainsDisplayOffsets` (page 183)
- `MediaDecodeTimeToSampleNum` (page 183)
- `MediaDisplayTimeToSampleNum` (page 184)
- `TrackTimeToMediaDisplayTime` (page 253)

There is additional support for programmers who work directly with arrays of media sample references. Although these new functions work with frame reordering video or other media with independent decode and display times, they can also be used with ordinary media types. See "QuickTime Sample Table API" (page 56).

Compressing Video Using Frame Reordering

When compressing video that uses frame reordering, there is no longer a one-to-one correspondence between submitting a frame for compression and getting back a compressed sample. The Image Compression Manager (ICM) and the compressor component may buffer multiple images before determining that a series of frames should be B-frames and a subsequent image should be decompressed out of order so that the B-frames can refer to it. The new ICM functions do not require a strict correlation between input frames and output frames. Frames may be rearranged by compression and decompression modules.

The new functions allow groups of multiple pixel buffers to be in use at various processing points in order to avoid unnecessary copying of data, using `CVPixelBuffers` and `CVPixelBufferPools`. These new types are Core Foundation based. They follow Core Foundation's protocols for reference counting (create/copy/retain/release). Each type has its own retain and release functions which are type-safe and NULL-safe, but otherwise equivalent to `CFRetain` and `CFRelease`. Note that the CVPixelBuffer functions generally provide their output data through callbacks, rather than as return values or function parameters.

In general, the new functions return `OSStatus`, with the exception of some simple Get functions that return single values.

Clients create compression sessions using `ICMCompressionSessionCreate` (page 106). They then feed pixel buffers in display order to `ICMCompressionSessionEncodeFrame` (page 108). Encoded frames may not be output immediately, and may not be returned in the same order as they are input—encoded frames will be returned in decode order, which will sometimes differ from display order. One of the parameters to `ICMCompressionSessionCreate` specifies a callback routine that QuickTime will call when each encoded frame is ready. Frames should be stored in the order they are output (decode order).

To force frames up to a certain display time to be encoded and output, call `ICMCompressionSessionCompleteFrames` (page 106).

To obtain a pixel buffer pool that satisfies the requirements of both your pixel buffer producer and the compressor, pass the pixel buffer producer's pixel buffer options into `ICMCompressionSessionCreate`, and then call `ICMCompressionSessionGetPixelBufferPool` (page 110). The compression session constructs an appropriate pixel buffer pool.

Alternatively, you can create your own pixel buffer pool by obtaining the compressor's pixel buffer attributes, choosing a format compatible with your pixel buffer producer, and setting that compressor's input format using the component properties API. The process of obtaining the pixel buffer attributes is illustrated in the following code snippet.

```
CFDictionaryRef attributesDictionary = NULL;

err = ICMCompressionSessionGetProperty(
    session,
    kQTPropertyClass_ICMCompressionSession,
    kICMCompressionSessionPropertyID_CompressorPixelBufferAttributes,
    sizeof(CFDictionaryRef),
    &attributesDictionary,
    NULL);

if (attributesDictionary) {
    // ...use...
    CFRelease(attributesDictionary);
}
```

You can also pass arbitrary pixel buffers to `ICMCompressionSessionEncodeFrame`; if they're incompatible with the compressor's requirements, then the compression session will make compatible copies and pass those to the compressor. This requires less setup but can result in significantly slower operation.

When the compressor no longer needs a source pixel buffer, it will release it. You may also pass `ICMCompressionSessionEncodeFrame` a callback to be called when the source pixel buffer is released.

Clients may call `ICMCompressionSessionGetImageDescription` (page 110) to get the image description for the encoded frames. Where possible, the ICM will allow this to be called before the first frame is encoded.

For additional details, see:

- `ICMCompressionSessionCreate` (page 106)
- `ICMCompressionSessionGetProperty` (page 111)
- `ICMCompressionSessionGetPixelBufferPool` (page 110)
- `ICMCompressionSessionEncodeFrame` (page 108)
- `ICMCompressionSessionGetImageDescription` (page 110)
- `ICMCompressionSessionCompleteFrames` (page 106)

H.264 Codec

The H.264 codec is the latest standards-based video codec. Published jointly by the ITU as H.264––Advanced Video Coding, and by ISO as MPEG-4 Part 10––Advanced Video Coding, the H.264 codec promises better image quality at lower bit rates than the current MPEG-4 video codec, and also better live streaming characteristics than the current H.263 codec.

This represents a significant increase in quality and performance, while operating in a standards-based framework.

QuickTime 7 for Mac OS X v10.4 includes a QuickTime decompressor component and an exporter component for creating and playing H.264-encoded video in QuickTime.

The H.264 codec makes use of QuickTime 7's new support for frame reordering video compression.

New Abstractions Layers For OpenGL Rendering

QuickTime 7 introduces the **visual context**—an abstraction that represents a visual output destination for a movie—and the **OpenGL texture context**, an implementation of the visual context that renders a movie's output as a series of OpenGL textures.

QuickTime Visual Context

A QuickTime visual context provides an abstraction layer that decouples QuickTime movies from GWorlds. This allows you to work in QuickTime without have to rely on QuickDraw concepts and structures. A visual context enables you to render QuickTime output using engines such as OpenGL.

A visual context can act as a virtual output device, rendering the movie's visual output, streaming it, storing it, or processing it in any number of ways.

A visual context can also act as a bridge between a QuickTime movie and an application's visual rendering environment. For example, you can set up a visual context for OpenGL textures. This causes a movie to produce its visual output as a series of OpenGL textures. You can then pass the textures to OpenGL for rendering, without having to copy the contents of a GWorld and transform it into an OpenGL texture yourself. In this case, the visual context performs the transformation from pixel buffers to OpenGL textures and delivers the visual output to your application.

A `QTVisualContextRef` is an opaque token that represents a drawing destination for a movie. The visual context is, in object-oriented terms, a base class for other concrete implementations of visual rendering environments. The output of the visual context depends entirely on the implementation. The implementation supplied with QuickTime 7 produces a series of OpenGL textures, but the list of possible outputs is extensible.

You create a visual context by calling a function that instantiates a context of a particular type, such as `QTOpenGLTextureContextCreate`. This allocates a context object suitable for passing to functions such as `SetMovieVisualContext` or `NewMovieFromProperties` (page 191), which target the visual output of the movie to the specified context.

Important: To use a visual context, open or create your movie using the `NewMovieFromProperties` function.

In order to use a visual context with a QuickTime movie, you should instantiate the movie using the new `NewMovieFromProperties` (page 191) function. This creates a movie that can accept a visual context. You can either specify the desired visual context when you instantiate the movie, or set the initial visual context to `NIL` (to prevent the movie from inheriting the current GWorld), and set the visual context later using `SetMovieVisualContext`. See "Replacing NewMovieFrom... Functions" (page 52) for details.

It is also possible to set a visual context for a movie that was instantiated using an older function, such as `NewMovieFromFile`. Such a movie will be associated with a GWorld. You change this to a visual context by first calling `SetMovieVisualContext` on the movie with the visual context set to `NIL`. This disassociates the movie from its GWorld (or any previous visual context). You can then call `SetMovieVisualContext` a second time, this time passing in a `QTVisualContextRef`.

OpenGL Texture Context

QuickTime 7 includes an OpenGL texture context.

A `QTOpenGLTextureContext` is a specific implementation of the visual context that provides the movie's visual data to a client application as OpenGL textures. These textures can then be rendered to the screen using OpenGL, composited with other graphics, run through CoreImage filters, or whatever else you, as the application developer, choose to do.

To use a `QTOpenGLTextureContext` for rendering to OpenGL, you must first set up an OpenGL session using your own code (Carbon, Cocoa, or CGL, for example).

Warning: Call QuickTime OpenGL texture context functions only when you are certain that no other thread is making calls to the same OpenGL context that the QuickTime texture context is using.

Follow these steps:

1. Create an OpenGL texture context by calling `QTOpenGLTextureContextCreate`, passing in the desired `CGLContext` and `CGLPixelFormat`.

2. Use the `QTVisualContextRef` that you get back to refer to your new visual context in other functions.

Note: Mac OS X v10.4 exposes a new function in Carbon's OpenGL API (AGL) to get a `CGLPixelFormat` from an `AGLPixelFormat`.

1. You can use the function `QTVisualContextSetImageAvailableCallback` to pass in an optional callback function if you want to be notified when the context has a new texture ready. A visual context callback of this type notifies you that a texture is available, but it may not actually be created until you ask for it. It is even possible for a texture to become invalid and be flushed after the callback and before the retrieval. Consequently, you should always poll for a new texture by calling `QTVisualContextIsNewImageAvailable` in your render loop.

2. The two functions that check for availability, `QTVisualContextNewImageAvailable` and the callback function, are the *only* QuickTime OpenGL texture context functions that are completely thread safe. All the other texture context functions must be called from a point in your application when it can be guaranteed that no other thread will make OpenGL calls to the same OpenGL context used by the texture context.

3. Set the `QTVisualContext` as the visual output of a movie by calling `NewMovieFromProperties` (page 191) or `SetMovieVisualContext`. Note that `SetMovieVisualContext` will fail if the movie was not opened using `NewMovieFromProperties`. Additionally, this call may fail if the host hardware is incapable of supporting the visual context for any reason. For example, many current graphics cards have a size limit of 2048 pixels in any dimension for OpenGL textures, so the attempt to set the visual context to OpenGL textures would fail with a larger movie. (One work-around for this problem is to resize the movie to fit within the hardware limitations given by `glGetIntegerv(GL_MAX_TEXTURE_SIZE, &maxTextureSize)`).

4. Poll for the availability of new textures by calling `QTVisualContextNewImageAvailable` during your render loop. Be aware that this function, or the optional callback, may be notifying you of a texture's availability well ahead of its display time, while previous undisplayed textures remain enqueued.

5. Your application can get a texture when you are ready to work with it by calling `QTVisualContextCopyImageForTime`. You may then pass the texture to OpenGL for display. Be aware, however, that calls to this function may produce a null texture at times when there is no visual output at the specified point in the movie timeline.

Important: You cannot call for textures out of order. Once you ask for a copy of the texture for a given time, that texture and any textures for previous times are no longer available.

1. You should make periodic calls to `QTVisualContextTask` during your program to allocate time for the OpenGL visual context to do its work.

2. Again, it is critical that this OpenGL texture function be called only at a point in your application when it can be guaranteed that no other thread will make OpenGL calls to the same OpenGL context used by the QuickTime visual context.

3. When you are done with the visual context, release it by calling `QTVisualContextRelease`. You can nest calls that retain and release the context as needed. These calls increment and decrement the context's reference count. When the count reachs zero, the context is deallocated and disposed. If your application creates the `QTOpenGLContext`, it is responsible for releasing it.

For additional details, see:

- QTVisualContextRef
- QTVisualContextCopyImageForTime (page 236)
- QTVisualContextGetAttribute (page 236)
- QTVisualContextGetTypeID (page 237)
- QTVisualContextIsNewImageAvailable (page 237)
- QTVisualContextSetAttribute (page 238)
- QTVisualContextSetImageAvailableCallback (page 239)
- QTVisualContextRetain (page 240)
- QTVisualContextRelease (page 239)
- QTVisualContextTask (page 240)
- SetMovieVisualContext
- GetMovieVisualContext
- QTOpenGLTextureContextCreate (page 210)
- QTVisualContextRetain
- QTVisualContextRelease
- QTOpenGLTextureAvailableCallbackProc (page 257)

Limitation Working With QuartzExtreme

Developers working on non-QuartzExtreme computers should be aware of a specific limitation with QTVisualContext. The limitation is that with QuartzExtreme turned off, the creation of a QTVisualContext fails with error -108.

If you launch QuartzDebug and turn off QuartzExtreme, and then launch the LiveVideoMixer and import a movie into a channel, it won't play. In the console, the following error message is returned: `QTVisualContext creation failed with error:-108`.

Replacing NewMovieFrom... Functions

QuickTime 7 introduces a replacement––`NewMovieFromProperties` (page 191)–– for `NewMovie`, `NewMovieFromDataRef`, and all other `NewMovieFrom...` functions.

In previous versions of QuickTime, you could use other functions that create new movies, including `NewMovieFromFile` and `NewMovieFromDataRef`. These functions accept flags that allow you to set some movie characteristics at creation time, but other movie characteristics are always set by default. For example, there must be a valid graphics port associated with a movie created by these functions, even if the movie does not output video.

`NewMovieFromProperties` (page 191) allows you to configure an extensible set of properties before creating a movie. This has a number of advantages.

- You can open a movie with exactly the properties you want, preventing QuickTime from taking undesired default actions.
- You can also specify properties that the older functions do not know about, such as a visual context for the movie.

Using NewMovieFromProperties

To instantiate a movie using `NewMovieFromProperties`, follow these steps:

1. Pass in a CFString file path, a URL, or set up a data reference, just as you would for `NewMovieFromDataRef` or one of the other `NewMovieFrom_` functions.
2. Next, set up a visual context for the movie to use by calling a function that creates a context of a the desired type, such as `QTOpenGLTextureContextCreate`. Similarly, you can set up an audio context if you want a device other than the default device.
3. Call `NewMovieFromProperties` (page 191), passing in the data reference for the movie and the `QTVisualContextRef` for the visual context, plus any appropriate properties listed in the `QTNewMoviePropertyArray`.

Properties are passed in using a `QTNewMoviePropertyElement` struct, which specifies the property class and property ID of each property.

The movie will automatically retain the visual context, so if your application does not need to work with the context directly, you may release it now.

For additional details, see:

- `QTNewMoviePropertyArray`
- `QTNewMoviePropertyElement`
- `NewMovieFromProperties` (page 191)

QuickTime Metadata Enhancements and API

QuickTime 7 introduces a new extensible metadata storage format that allows more flexible and efficient storage of metadata, including encapsulated storage of metadata in native format (without translating to and from a defined QuickTime format). For developers, this means that you can now write cleaner, more generic code that enables you to look at, for example, all the metadata in a QuickTime or iTunes music track using just a single function call.

Metadata, of course, is information about a file, track, or media, such as the white balance used to create a photographic image or the artist, album, and title of an MP3 track. Traditionally, metadata information is stored in QuickTime user data items or in ancilliary tracks in a movie. For example, copyright information is normally stored in a `'@cpy'` user data item, while cover art for an AAC audio track is normally stored in a track that is not displayed by all applications.

The new metadata enhancements in QuickTime 7 allow you to access both old (QuickTime) and new (iTunes) formats. The new metadata storage format is intended as a replacement for QuickTime user data, which was limited in features and robustness. Specifically, the metadata enhancements introduced in QuickTime 7 provide the following capabilities:

- The ability to assign data types to any metadata that you place into the new storage format.
- The ability to assign locale information––for example, a particular language or country––to the format.
- The ability to use a more descriptive key in the storage format––for example, a reverse DNS format, such as com.apple.quicktime.mov.

The new metadata format allows storage of data that is not possible in user data items, without extending the list of item types exhaustively, and allows labeling of metadata unambiguously, rather than as data in an undisplayed media track. It also permits inclusion of metadata that is always stored external to the movie, even when the movie is flattened (saved as self-contained).

How It Works

Metadata is encapsulated in an opaque container and accessed using a `QTMetDataRef`. A `QTMetaDataRef` represents a metadata repository consisting of one or more native metadata containers. The QuickTime metadata API supports unified access to and management of these containers.

Each container consists of some number of metadata items. Metadata items correspond to individually labeled values with characteristics such as keys, data types, locale information, and so on. Note that what QuickTime calls *items* are sometimes referred to as *attributes* or *properties* in other metadata systems.

You address each container by its storage format (`kQTMetaDataStorageFormat`). Initially, there is support for classic QuickTime user data items, iTunes metadata, and a richer QuickTime metadata container format. A `QTMetaDataRef` may have one or all of these. No direct access to the native storage containers is provided.

`QTMetaDataRefs` may be associated with a movie, track or media. This parallels user data atoms usage but provides access to other kinds of metadata storage at those levels.

A metadata item is assigned a runtime identifier (`QTMetaDataItem`) that along with the `QTMetaDataRef` identifies the particular item (and value) across all native containers managed by the `QTMetaDataRef`.

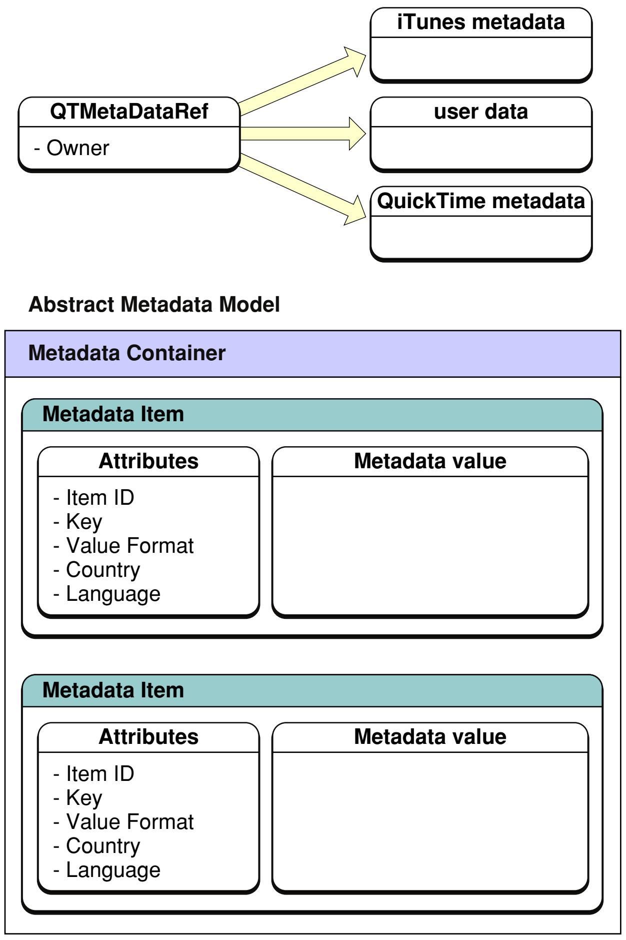
Each item is addressed by a key, or label. The key is not necessarily unique within its container, as it is possible to have multiple items with the same key (for example, multiple author items). Functions exist to enumerate all items or only items with a particular key.

Because a `QTMetaDataRef` may provide access to different native metadata containers with differing key structures—a four-char-code for one, a string for another, and so on—the key structure is also specified. A `QTMetaDataKeyFormat` indicates the key structure to functions that take keys. This also supports container formats that allow multiple key structures or multiple versions of key structures.

To allow unified access across disparate containers, you can specify a wildcard storage format. This can be used for operations such as searches across container formats. A special key format called `kQTMetaDataKeyFormatCommon` indicates one of a set of common keys that can be handled by multiple native containers (for example, copyright).

Both modes of operation are illustrated in Figure 2-37.

Figure 2-37 Metadata modes of operations



Advantages of the New Metadata Format

The QuickTime metadata format is inherently extensible. Instead of a set of structures and enumerated parameters, the metadata API uses a set of properties that can be enumerated, and whose characteristics can be discovered, dynamically at runtime. This is analogous to the QuickTime component property function in that you first get the property info, such as its size and format, and then you allocate the appropriate container or structure to get or set the actual property.

The new QuickTime metadata format and API consist of the following structures, enumerations, and functions, grouped in sections followed by the specific functions:

- "`Metadata Format Constants`" (page 283)
- "`Metadata Property IDs`" (page 283)
- "`Metadata Key Constants`" (page 284)
- "`Metadata Error Codes`" (page 285)
- `QTCopyMovieMetaData` (page 194)
- `QTCopyTrackMetaData` (page 195)

- `QTCopyMediaMetaData` (page 196)
- `QTMetaDataAddItem` (page 198)
- `QTMetaDataGetItemCount` (page 199)
- `QTMetaDataGetItemProperty` (page 200)
- `QTMetaDataGetItemPropertyInfo` (page 201)
- `QTMetaDataGetItemValue` (page 202)
- `QTMetaDataGetNextItem` (page 203)
- `QTMetaDataGetProperty` (page 204)
- `QTMetaDataGetPropertyInfo` (page 205)
- `QTMetaDataRetain` (page 205)
- `QTMetaDataRelease` (page 206)
- `QTMetaDataRemoveItem` (page 206)
- `QTMetaDataRemoveItemsWithKey` (page 207)
- `QTMetaDataSetItem` (page 208)
- `QTMetaDataSetItemProperty` (page 209)
- `QTMetaDataSetProperty` (page 210)

QuickTime Sample Table API

The new QuickTime sample table API in QuickTime 7 is used when you need to obtain information about samples—such as their size, location, and sample descriptions—or to set this kind of information (for example, when adding samples or blocks of samples to a media directly, without using the services of an importer or sequence grabber).

This new API introduces `QTSampleTable` as a logical replacement for the arrays of sample reference records used with the older functions `AddMediaSampleReferences` and `AddMediaSampleReferences64`. New functions allow you to operate on whole tables of data simultaneously.

Like many new QuickTime APIs, the QuickTime sample table API uses opaque data types whose properties can be discovered dynamically at runtime. This is analogous to the component properties API for configuring components. You use a `GetPropertyInfo` function to discover the size and format of a property, then allocate the necessary container or structure to get or set the actual property.

This API works with both simple media types that have a single media time for each sample and new media types such as frame reordering video that have independent decode and display times for samples.

Warning: When using the QuickTime sample table API to work with constant-bit-rate (CBR) compressed audio, the audio is represented in a new way.

In the QuickTime sample table API, sample numbers for audio always refer to packets. This is simpler and more consistant, but it means that a new function may not return the same value as an older, analogous function when called with reference to compressed CBR sound. For example, `QTSampleTableGetNumberOfSamples` may return a different sample count than `GetMediaSampleCount`.

All compressed audio is quantized into packets, and each packet can be decompressed into multiple PCM samples. With previous APIs, media sample numbers for CBR sound refer to PCM samples, rather than the compressed packets. When the same APIs are applied to variable-bit-rate (VBR) sound, however, the sample numbers refer to packets. This inconsistency means that code using these older APIs must handle CBR and VBR differently. In this API, by contrast, sample numbers always refer to packets.

This applies only to *compressed* CBR sound, however. In uncompressed sound tracks, each packet is simply an uncompressed PCM frame, so the value is the same whether the sample number refers to packets or PCM samples.

For full details of the QuickTime sample table API, see:

- `QTSampleTableCreateMutable` (page 215)
- `QTSampleTableCreateMutableCopy` (page 216)
- `QTSampleTableAddSampleDescription` (page 212)
- `QTSampleTableCopySampleDescription` (page 214)
- `QTSampleTableAddSampleReferences` (page 213)
- `AddSampleTableToMedia` (page 66)
- `CopyMediaMutableSampleTable` (page 67)
- `QTSampleTableReplaceRange` (page 226)
- `QTSampleTableGetProperty` (page 220)
- `QTSampleTableSetProperty` (page 227)
- `QTSampleTableGetPropertyInfo` (page 221)
- `QTSampleTableGetNumberOfSamples` (page 220)
- `QTSampleTableGetSampleDescriptionID` (page 223)
- `QTSampleTableGetDataSizePerSample` (page 217)
- `QTSampleTableGetSampleFlags` (page 223)
- `QTSampleTableGetDataOffset` (page 216)
- `QTSampleTableGetDisplayOffset` (page 218)
- `QTSampleTableGetTypeID` (page 225)
- `QTSampleTableGetDecodeDuration` (page 217)
- `QTSampleTableGetNextAttributeChange` (page 218)
- `QTSampleTableGetTimeScale` (page 224)
- `QTSampleTableRelease` (page 225)
- `QTSampleTableReplaceRange` (page 226)
- `QTSampleTableRetain` (page 226)
- `QTSampleTableSetTimeScale` (page 228)

JavaScript Support and Accessibility in Safari

New in QuickTime 7, the QuickTime plug-in for Safari is now fully scriptable using JavaScript. This means you can now use JavaScript to control QuickTime when webpages are viewed using Safari.

Two plug-ins are available in QuickTime 7 for Mac OS X v10.4: Carbon and a new Cocoa plug-in. From the user's perspective, these plug-ins look and behave the same. The Cocoa plug-in works only in Safari, however. The benefit for both users and developers is that the plug-in is now scriptable.

To control a movie through the QuickTime plug-in using JavaScript, you must include the parameter `EnableJavaSript="true"` in the movie's `EMBED` tag (this parameter is not needed in the `OBJECT` tag, but it does no harm there).

JavaScript treats each embedded QuickTime movie in a webpage as a separately addressable object. Movies can be identified by name if there is a `NAME` parameter in the movie's `EMBED` tag and an `ID` attribute in the movie's `OBJECT` tag. Internet Explorer for Windows uses the `ID` attribute. Other browsers use the `NAME` parameter. Both `NAME` and `ID` should be set to the same value.

For example, to create a movie that can be addressed in JavaScript as `Movie1`, your `OBJECT` and `EMBED` tags would look something like this:

```
<OBJECT classid="clsid:02BF25D5-8C17-4B23-BC80-D3488ABDDC6B"
    codebase="http://www.apple.com/qtactivex/qtplugin.cab"
    width="180" height="160"
    id="movie1" >

    <PARAM name="src" value="My.mov">

    <EMBED width="180" height="160"
        src="My.mov"
        name="movie1"
        enablejavascript="true">
    </EMBED>
</OBJECT>
```

Movies can also be identified by their ordinal number in the JavaScript `embeds[]` array.

An example of usage and syntax, showing JavaScript control of multiple QuickTime movies using different methods of addressing, can be found in Sample JavaScript Usage.

QuickTime exposes dozens of methods to JavaScript, allowing you to control not only the standard user interface actions, such as playing and stopping a movie, but also more complex actions, such as layering and compositing. You can use JavaScript, for example, to enable and disable alternate audio, text, or video tracks, or change a video track's graphics mode or a sprite's current image.

Detailed descriptions of the QuickTime methods and properties available to JavaScript can be found in JavaScript Support.

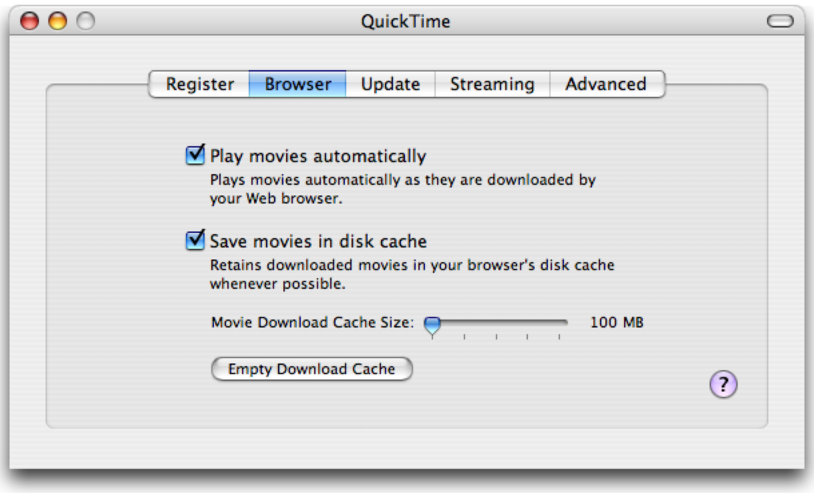
Other Changes and Enhancements

This section discusses the following changes and enhancements that are available in QuickTime 7.

New Persistent Cache Option

QuickTime 7 introduces a new **persistent cache** option, which is enabled in the System Preferences > QuickTime > Browser panel, as shown in Figure 2-38. Users, web authors, and web content developers should understand the consequences of this new option because it may impact the way that QuickTime content is downloaded and saved from their websites. The reason is that QuickTime's caching behavior has changed in this release.

Figure 2-38 QuickTime Browser preferences with the Save movies in disk cache box checked



☑ Play movies automatically
Plays movies automatically as they are downloaded by your Web browser.

☑ Save movies in disk cache
Retains downloaded movies in your browser's disk cache whenever possible.

Movie Download Cache Size: 100 MB

Important: The new cache is *only* used by QuickTime. Content downloaded by QuickTime is *not* stored in the browser cache.

Changes to File Caching

By default, the user preference is set to "Save movies in disk cache." This means that files downloaded by QuickTime and written to the cache will now stay in the cache when the last connection to the file is closed. By contrast, in pre-QuickTime 7 versions, downloaded files would be written to disk and but only remain there as long as the user kept open a connection to the file. After the movie was closed, the local file would be deleted. If you wanted to look at the movie again, you would have to download the file in its entirety another time.

If the user unchecks the preference, QuickTime's old behavior prevails—that is, movies downloaded to disk will only be saved as long as they remain open. There may be situations, for example, when unchecking the box is warranted: users don't want any QuickTime content to be cached perhaps for privacy reasons, or if they don't have enough disk space on their computers.

When the preference is checked, movies downloaded by QuickTime will be saved in the cache. The slider allows the user to set the maximum size of the cache (the minimum is 100 MB). Note that QuickTime's cache is per user, not computer wide. This means that if you have more than one account on the computer, each user's setting and cached movies are for themselves only.

When the cache is on, files may not remain in the persistent cache because of settings on the server. If you have administrator (admin) access to a server, you can tell the server to include information in the file header that specifies how long it is allowed to stay on the user's computer. The header can say, "Don't cache this at all." Or, "Only keep it for a week." QuickTime pays close attention to the information that is stored in the headers.

In particular, when the cache is enabled QuickTime honors the "expiration date," often supplied by web servers to inform clients of how long a file remains valid after it has been downloaded. Until the expiration date supplied by the server is reached, QuickTime will respond to additional requests for the file by supplying the file from the cache instead of by fetching it again from the server. This remains true even if the file is changed on the server before the expiration date is reached. Previous versions of QuickTime as described above always downloaded a file afresh, if a previous version of it was no longer still open and in use, and therefore changes to files made before the expiration date would often be accessible immediately.

How To Control File Caching

There are several ways to control QuickTime's file caching behavior. Each step, listed below, may be appropriate for a different situation:

- **HTTP headers**. Useful only if you have admin access to the server on which the files are posted. Consult your web server documentation for specifics (for example, look for "Cache-Control: no-cache" and "Expires:").
- **"cache" EMBED/OBJECT attribute**. Useful if your movies are embedded in HTML pages, but you don't have adminaccess to the server. This is a very common situation.

  In previous versions of QuickTime, this tag told the plug-in to ask the browser not to keep an embedded movie in its cache. This is still true, but now it also tells the plug-in to not keep the movie in the QuickTime persistent cache either.
- **XML movies**. The "cache" attribute has been added to QuickTime Media Link importer. Adding 'cache="false"' to a media link movie will keep it from being saved in QuickTime's persistent cache. See the documentation QuickTime Media Links XML Importer for more information about this movie importer.

When a movie is tagged as "not cacheable" with any of these methods, QuickTime will not keep the movie or any media loaded by it in the persistent cache (for example, sprite images loaded by URL).

Updates to QuickTime for Java

QuickTime for Java (QTJ) is now fully supported in QuickTime 7. QTJ is now installed by default in QuickTime 7.

This release includes a number of important bug fixes requested by QuickTime for Java developers. These are as follows:

- Major fixes for issues related to drawing and correct QTComponent rendering
- Compatibility with headless applications
- Fixes for issues related to movie progress procedures and movie exporting
- Fixes for Applet issues, including support for MPEG video playback in an applet
- Support for 2-byte character file names

- Security fixes

Support for Quartz Composer

QuickTime can read Quartz Composers. Also, Quartz Composer compositions can be exported as QuickTime movies.

You can accomplish this either by using the Export menu command in the Quartz Composer Editor, or by opening the composition in QuickTime Player and saving it as a movie.

New Functions, Data Types, and Constants in QuickTime 7

This chapter describes all the new QuickTime functions, data structures, constants, and callbacks available in this software release.

If you are a QuickTime API-level developer, content author, multimedia producer or Webmaster who is currently working with QuickTime, you should read this chapter and use it for reference, as needed, in your application development.

QuickTime 7 API Reference

The following functions, callbacks, structures, and constants are new or changed with this release of QuickTime.

Functions

Functions that are new in QuickTime 7 are described in this section; they are listed in alphabetical order.

AddMediaSample2

Adds sample data and a description to a media.

```
OSErr AddMediaSample2 (
 Media                      theMedia,
 const UInt8                *dataIn,
 ByteCount                  size,
 TimeValue64                decodeDurationPerSample,
 TimeValue64                displayOffset,
 SampleDescriptionHandle    sampleDescriptionH,
 ItemCount                  numberOfSamples,
 MediaSampleFlags           sampleFlags,
 TimeValue64                *sampleDecodeTimeOut );
```

Parameters

theMedia

The media for this operation. You obtain this media identifier from such functions as `NewTrackMedia` and `GetTrackMedia`.

dataIn

A handle to the sample data. The function adds this data to the media specified by *theMedia*. You specify the number of bytes of sample data with the *size* parameter.

size

The number of bytes of sample data to be added to the media. This parameter indicates the total number of bytes in the sample data to be added to the media, not the number of bytes per sample. Use the *numberOfSamples* parameter to indicate the number of samples that are contained in the sample data.

decodeDurationPerSample

The duration of each sample to be added, representing the amount of time that passes while the sample data is being displayed. You must specify this parameter in the media's time scale. For example, if you are adding sound that was sampled at 22 kHz to a media that contains a sound track with the same time scale, you would set *durationPerSample* to 1. Similarly, if you are adding video that was recorded at 10 frames per second to a video media that has a time scale of 600, you would set this parameter to 60. Note that this is the duration *per sample*, regardless of the number of samples being added.

displayOffset

A 64-bit time value that specifies the offset between the decode time (the start time of the track plus the duration of all previous samples) and the display time. This value is normally zero unless the sample is frame reordering compressed video.

sampleDescriptionH

A handle to a `SampleDescription` structure. Some media structures may require sample descriptions. There are different descriptions for different types of samples. For example, a media that contains compressed video requires that you supply an `ImageDescription` structure. A media that contains sound requires that you supply a `SoundDescription` structure. If the media does not require a `SampleDescription` structure, set this parameter to NIL.

numberOfSamples

The number of samples contained in the sample data to be added to the media. The Movie Toolbox considers the value of this parameter as well as the value of the *size* parameter when it determines the size of each sample that it adds to the media. You should set the value of this parameter so that the resulting sample size represents a reasonable compromise between total data retrieval time and the overhead associated with input and output. You should also consider the speed of the data storage device; CD-ROM devices are much slower than hard disks, for example, and should therefore have a smaller sample size. For a video media, set a sample size that corresponds to the size of a frame. For a sound media, choose a number of samples that corresponds to between 0.5 and 1.0 seconds of sound. In general, you should not create groups of sound samples that are less than 2 KB in size or greater than 15 KB. Typically, a sample size of about 8 KB is reasonable for most storage devices.

sampleFlags

Flags that control the add operation; set unused flags to 0:

`mediaSampleNotSync`

Indicates that the sample to be added is not a sync sample. Set this flag to 1 if the sample is not a sync sample; set it to 0 if the sample is a sync sample.

sampleDecodeTimeOut

A pointer to a time value that represents the sample decode time. After adding the sample data to the media, the function returns in this parameter the time where the sample was inserted. If you don't want to receive this information, set this parameter to `NIL`.

Return Value

An error code. Returns `noErr` if there is no error. You can access Movie Toolbox error returns through `GetMoviesError` and `GetMoviesStickyError`, as well as in the function result.

Discussion

Your application specifies the sample and the media for the operation. This function updates the media so that it contains the sample data. One call to this function can add several samples to a media. This function replaces `AddMediaSample`; it adds 64-bit support and support for frame reordering video compression (display offset). This function can return these errors:

`noErr`
: Success.

`memFullErr`
: Could not allocate memory.

`paramErr`
: Invalid parameter.

`errMediaDoesNotSupportDisplayOffsets`
: The media does not support nonzero display offsets.

`errDisplayTimeAlreadyInUse`
: There is already a sample with this display time.

`errDisplayTimeTooEarly`
: A sample's display time would be earlier than the display time of an existing sample that does not have the `mediaSampleEarlierDisplayTimesAllowed` flag set.

Version Notes

Introduced in QuickTime 7. This function extends and supersedes `AddMediaSample`. Whereas `AddMediaSample` takes a `Handle+offset+size`, `AddMediaSample2` takes a `Ptr+size`.

Availability

Carbon status: Supported C interface file: `Movies.h`

Declared In

`Movies.h`

AddMediaSampleFromEncodedFrame

Adds sample data and description from an encoded frame to a media.

```
OSErr AddMediaSampleFromEncodedFrame (
 Media theMedia,
 ICMEncodedFrameRef encodedFrame,
 TimeValue64 *sampleDecodeTimeOut );
```

Parameters

`theMedia`
: The media for this operation. You obtain this media identifier from such functions as `NewTrackMedia` and `GetTrackMedia`

`encodedFrame`
: An encoded frame token returned by an ICMCompressionSequence.

`sampleDecodeTimeOut`
: A pointer to a time value. After adding the sample data to the media, the function returns the decode time where the first sample was inserted in the time value referred to by this parameter. If you don't want to receive this information, set this parameter to `NULL`.

Return Value
An error code. Returns `noErr` if there is no error. You can access Movie Toolbox error returns through `GetMoviesError` and `GetMoviesStickyError`, as well as in the function result.

Discussion
This is a convenience API to make it easy to add frames emitted by new ICM compression functions to media. It can return these errors:

`noErr`
    Success.

`memFullErr`
    Could not allocate memory.

`paramErr`
    Invalid parameter.

`errMediaDoesNotSupportDisplayOffsets`
    The media does not support nonzero display offsets.

`errDisplayTimeAlreadyInUse`
    There is already a sample with this display time.

`errDisplayTimeTooEarly`
    A sample's display time would be earlier than the display time of an existing sample that does not have the `mediaSampleEarlierDisplayTimesAllowed` flag set.

Version Notes
Introduced in QuickTime 7

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

AddSampleTableToMedia

Adds a sample table to a media.

```
OSErr AddSampleTableToMedia (
 Media                theMedia,
 QTSampleTableRef     sampleTable,
 SInt64               startSampleNum,
 SInt64               numberOfSamples,
 TimeValue64          *sampleDecodeTimeOut );
```

Parameters

`theMedia`
    The media for this operation. You obtain this media identifier from such functions as `NewTrackMedia` and `GetTrackMedia`.

`sampleTable`
    A reference to an opaque sample table object containing sample references to be added to the media.

`startSampleNum`
    The sample number of the first sample reference in the sample table to be added to the media. The first sample's number is 1.

numberOfSamples

The number of sample references from the sample table to be added to the media.

sampleDecodeTimeOut

A pointer to a time value. After adding the sample references to the media, the function returns the decode time where the first sample was inserted in the time value referred to by this parameter. If you don't want to receive this information, set this parameter to `NULL`.

Return Value

An error code. Returns `noErr` if there is no error. You can access Movie Toolbox error returns through `GetMoviesError` and `GetMoviesStickyError`, as well as in the function result.

Discussion

This function can return these errors:

`noErr`

Success.

`memFullErr`

Could not allocate memory.

`paramErr`

Invalid parameter.

`errMediaDoesNotSupportDisplayOffsets`

The media does not support nonzero display offsets.

`errDisplayTimeAlreadyInUse`

There is already a sample with this display time.

`errDisplayTimeTooEarly`

A sample's display time would be earlier than the display time of an existing sample that does not have the `mediaSampleEarlierDisplayTimesAllowed` flag set.

If `errDisplayTimeAlreadyInUse` or `errDisplayTimeTooEarly` is returned, no samples are added.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `Movies.h`

Declared In

`Movies.h`

CopyMediaMutableSampleTable

Obtains information about sample references in a media in the form of a sample table.

```
OSErr CopyMediaMutableSampleTable (
 Media                        theMedia,
 TimeValue64                  startDecodeTime,
 TimeValue64                  *sampleStartDecodeTime,
 SInt64                       maxNumberOfSamples,
 TimeValue64                  maxDecodeDuration,
 QTMutableSampleTableRef      *sampleTableOut );
```

Parameters

theMedia
: The media for this operation. You obtain this media identifier from such functions as `NewTrackMedia` and `GetTrackMedia`.

startDecodeTime
: A 64-bit time value that represents the starting decode time of the sample references to be retrieved. You must specify this value in the media's time scale.

sampleStartDecodeTime
: A pointer to a time value. The function updates this time value to indicate the actual decode time of the first returned sample reference. If you are not interested in this information, set this parameter to `NULL`. The returned time may differ from the time you specified with the *startDecodeTime* parameter. This will occur if the time you specified falls in the middle of a sample.

maxNumberOfSamples
: A 64-bit signed integer that contains the maximum number of sample references to be returned. If you set this parameter to 0, the Movie Toolbox uses a value that is appropriate to the media.

maxDecodeDuration
: A 64-bit time value that represents the maximum decode duration to be returned. The function does not return samples with greater decode duration than you specify with this parameter. If you set this parameter to 0, the Movie Toolbox uses a value that is appropriate for the media.

sampleTableOut
: A reference to an opaque sample table object. When you are done with the returned sample table, release it with `QTSampleTableRelease`.

Return Value
An error code. Returns `memFullErr` if it could not allocate memory, `paramErr` if there was an invalid parameter, or `noErr` if there is no error. You can access Movie Toolbox error returns through `GetMoviesError` and `GetMoviesStickyError`, as well as in the function result.

Discussion
To find out how many samples were returned in the sample table, call `QTSampleTableGetNumberOfSamples`.

Version Notes
Introduced in QuickTime 7. This function supersedes `GetMediaSampleReferences` and `GetMediaSampleReferences64`.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

DisposeMovieExportStageReachedCallbackUPP

Disposes of a `MovieExportStageReachedCallbackUPP` pointer.

```
void DisposeMovieExportStageReachedCallbackUPP (
 MovieExportStageReachedCallbackUPP    userUPP );
```

Parameters

userUPP

A `MovieExportStageReachedCallbackUPP` pointer.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `QuickTimeComponents.h`

Declared In

`QuickTimeComponents.h`

DisposeQTTrackPropertyListenerUPP

Disposes a track property listener UPP.

```
void DisposeQTTrackPropertyListenerUPP (
 QTTrackPropertyListenerUPP userUPP );
```

Parameters

userUPP

A QTTrackPropertyListenerUPP pointer. See Universal Procedure Pointers in the `QuickTime API Reference` for more information.

Version Notes

Introduced in QuickTime 7

Availability

Carbon status: Supported C interface file: `Movies.h`

Declared In

`Movies.h`

ExtendMediaDecodeDurationToDisplayEndTime

Prepares a media for the addition of a completely new sequence of samples by ensuring that the media display end time is not later than the media decode end time.

```
OSErr ExtendMediaDecodeDurationToDisplayEndTime (
 Media      theMedia,
 Boolean    *mediaChanged );
```

Parameters

theMedia

The media for this operation. You obtain this media identifier from such functions as `NewTrackMedia` and `GetTrackMedia`.

mediaChanged

A pointer to a `Boolean` that returns `TRUE` if any samples in the media were adjusted, `FALSE` otherwise. If you don't want to receive this information, set this parameter to `NULL`.

Return Value
An error code. Returns `memFullErr` if it could not allocate memory, `paramErr` if there was an invalid parameter, or `noErr` if there is no error. You can access Movie Toolbox error returns through `GetMoviesError` and `GetMoviesStickyError`, as well as in the function result.

Discussion
After adding a complete, well-formed set of samples to a media, the media's display end time should be the same as the media's decode end time (also called the media decode duration). However, this is not necessarily the case after individual sample-adding operations, and hence it is possible for a media to be left with a display end time later than its decode end time (if adding a sequence of frames is aborted halfway, for example).

This may make it difficult to add a new group of samples, because a well-formed group of samples' earliest display time should be the same as the first frame's decode time. If such a well-formed group is added to an incompletely finished media, frames from the old and new groups frames might collide in display time.

This function prevents any such collision or overlap by extending the last sample's decode duration as necessary. It ensures that the next added sample will have a decode time no earlier than the media's display end time. If this was already the case, it makes no change to the media.

You can call this function before you begin adding samples to a media if you're not certain that the media was left in a well-finished state. You do not need to call it before adding samples to a newly created media, nor should you call it between sample additions from the same compression session.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

GetDSequenceNonScheduledDisplayDirection

Returns the display direction for a decompress sequence.

```
OSErr GetDSequenceNonScheduledDisplayDirection (
 ImageSequence    sequence,
 Fixed            *rate );
```

Parameters

`sequence`
Contains the unique sequence identifier that was returned by the `DecompressSequenceBegin` function.

`rate`
A pointer to the display direction. Negative values represent backward display and positive values represent forward display.

Return Value
An error code. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

GetDSequenceNonScheduledDisplayTime

Gets the display time for a decompression sequence.

```
OSErr GetDSequenceNonScheduledDisplayTime (
 ImageSequence    sequence,
 TimeValue64      *displayTime,
 TimeScale        *displayTimeScale );
```

Parameters

sequence
: Contains the unique sequence identifier that was returned by the `DecompressSequenceBegin` function.

displayTime
: A pointer to a variable to hold the display time.

displayTimeScale
: A pointer to a variable to hold the display time scale.

Return Value
An error code. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

GetMediaAdvanceDecodeTime

Returns the advance decode time of a media.

```
TimeValue64 GetMediaAdvanceDecodeTime (
 Media    theMedia );
```

Parameters

theMedia
: The media for this operation. You obtain this media identifier from such functions as `NewTrackMedia` and `GetTrackMedia`.

Return Value
A 64-bit time value that represents the media's advance decode time. A media's advance decode time is the absolute value of the greatest-magnitude negative display offset of its samples, or 0 if there are no samples with negative display offsets. This is the amount that the decode time axis must be adjusted ahead of the display time axis to ensure that no sample's adjusted decode time is later than its display time. For media without nonzero display offsets, the advance decode time is 0.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

GetMediaDataSizeTime64

Determines the size, in bytes, of the sample data in a media segment.

```
OSErr GetMediaDataSizeTime64 (
 Media          theMedia,
 TimeValue64    startDisplayTime,
 TimeValue64    displayDuration,
 SInt64         *dataSize );
```

Parameters

theMedia
The media for this operation. You obtain this media identifier from such functions as `NewTrackMedia` and `GetTrackMedia`.

startDisplayTime
A 64-bit time value that specifies the starting point of the segment in media display time.

displayDuration
A 64-bit time value that specifies the duration of the segment in media display time.

dataSize
A pointer to a variable to receive the size, in bytes, of the sample data in the defined media segment.

Return Value
An error code. Returns `noErr` if there is no error. You can access Movie Toolbox error returns through `GetMoviesError` and `GetMoviesStickyError`, as well as in the function result.

Discussion
The only difference between this function and `GetMediaDataSize64` is that this function uses 64-bit time values and returns a 64-bit size.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

GetMediaDecodeDuration

Returns the decode duration of a media.

```
TimeValue64 GetMediaDecodeDuration (
 Media    theMedia );
```

Parameters

theMedia

The media for this operation. You obtain this media identifier from such functions as `NewTrackMedia` and `GetTrackMedia`.

Return Value

A 64-bit time value that repre sents the media's decode duration. A media's decode duration is the sum of the decode durations of its samples.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `Movies.h`

Declared In

`Movies.h`

GetMediaDisplayDuration

Returns the display duration of a media.

```
TimeValue64 GetMediaDisplayDuration (
 Media    theMedia );
```

Parameters

theMedia

The media for this operation. You obtain this media identifier from such functions as `NewTrackMedia` and `GetTrackMedia`.

Return Value

A 64-bit time value that represents the media's display duration. A media's display duration is its display end time minus its display start time. For media without nonzero display offsets, the decode duration and display duration are the same.

Discussion

When inserting media with display offsets into a track, use display time:

```
InsertMediaIntoTrack( track,
                0,                                // track start time
                GetMediaDisplayStartTime( media ),   // media start time
                GetMediaDisplayDuration( media ),
                fixed1 );
```

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

GetMediaDisplayEndTime

Returns the display end time of a media.

```
TimeValue64 GetMediaDisplayEndTime (
 Media    theMedia );
```

Parameters

theMedia
: The media for this operation. You obtain this media identifier from such functions as `NewTrackMedia` and `GetTrackMedia`.

Return Value
A 64-bit time value that represents the media's display end time. A media's display end time is the sum of the display time and decode duration of the sample with the greatest display time. For media without nonzero display offsets, the display end time is the same as the media's decode duration.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

GetMediaDisplayStartTime

Returns the display start time of a media.

```
TimeValue64 GetMediaDisplayStartTime (
 Media    theMedia );
```

Parameters

theMedia
: The media for this operation. You obtain this media identifier from such functions as `NewTrackMedia` and `GetTrackMedia`.

Return Value
A 64-bit time value that represents the media's display start time. A media's display start time is the earliest display time of any of its samples. For media without nonzero display offsets, the display start time is always 0.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

GetMediaNextInterestingDecodeTime

Searches for decode times of interest in a media.

```
void GetMediaNextInterestingDecodeTime (
 Media          theMedia,
 short          interestingTimeFlags,
 TimeValue64    decodeTime,
 Fixed          rate,
 TimeValue64    *interestingDecodeTime,
 TimeValue64    *interestingDecodeDuration );
```

Parameters

theMedia

The media for this operation. You obtain this media identifier from such functions as `NewTrackMedia` and `GetTrackMedia`.

interestingTimeFlags

Flags that determine the search criteria. Note that you may set only one of the `nextTimeMediaSample`, `nextTimeMediaEdit`, or `nextTimeSyncSample` flags to 1. Set unused flags to 0:

`nextTimeMediaSample`

Set this flag to 1 to search for the next sample.

`nextTimeMediaEdit`

Set this flag to 1 to search for the next group of samples.

`nextTimeSyncSample`

Set this flag to 1 to search for the next sync sample.

`nextTimeEdgeOK`

Set this flag to 1 to accept information about elements that begin or end at the time specified by the *decodeTime* parameter. When this flag is set the function returns valid information about the beginning and end of a media.

decodeTime

Specifies the starting point for the search in decode time. This time value must be expressed in the media's time scale.

rate

The search direction. Negative values cause the Movie Toolbox to search backward from the starting point specified in the *time* parameter. Other values cause a forward search.

interestingDecodeTime

On return, a pointer to a 64-bit time value in decode time. The Movie Toolbox returns the first time value it finds that meets the search criteria specified in the *flags* parameter. This time value is in the media's time scale. If there are no times that meet the search criteria you specify, the Movie Toolbox sets this value to –1. Set this parameter to `NULL` if you are not interested in this information.

interestingDecodeDuration

On return, a pointer to a 64-bit time value in decode time. The Movie Toolbox returns the duration of the interesting time in the media's time coordinate system. Set this parameter to `NULL` if you don't want this information; this lets the function works faster.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

GetMediaNextInterestingDisplayTime

Searches for display times of interest in a media.

```
void GetMediaNextInterestingDisplayTime (
 Media          theMedia,
 short          interestingTimeFlags,
 TimeValue64    displayTime,
 Fixed          rate,
 TimeValue64    *interestingDisplayTime,
 TimeValue64    *interestingDisplayDuration );
```

Parameters

theMedia
: The media for this operation. You obtain this media identifier from such functions as `NewTrackMedia` and `GetTrackMedia`.

interestingTimeFlags
: Flags that determine the search criteria. Note that you may set only one of the `nextTimeMediaSample`, `nextTimeMediaEdit`, or `nextTimeSyncSample` flags to 1. Set unused flags to 0:

  `nextTimeMediaSample`
  : Set this flag to 1 to search for the next sample.

  `nextTimeMediaEdit`
  : Set this flag to 1 to search for the next group of samples.

  `nextTimeSyncSample`
  : Set this flag to 1 to search for the next sync sample.

  `nextTimeEdgeOK`
  : Set this flag to 1 to accept information about elements that begin or end at the time specified by the *decodeTime* parameter. When this flag is set the function returns valid information about the beginning and end of a media.

displayTime
: Specifies the starting point for the search in display time. This time value must be expressed in the media's time scale.

rate
: The search direction. Negative values cause the Movie Toolbox to search backward from the starting point specified in the *time* parameter. Other values cause a forward search.

interestingDisplayTime
: On return, a pointer to a 64-bit time value in display time. The Movie Toolbox returns the first time value it finds that meets the search criteria specified in the *flags* parameter. This time value is in the media's time scale. If there are no times that meet the search criteria you specify, the Movie Toolbox sets this value to –1. Set this parameter to `NIL` if you are not interested in this information.

interestingDisplayDuration

On return, a pointer to a 64-bit time value in display time. The Movie Toolbox returns the duration of the interesting time in the media's time coordinate system. Set this parameter to `NIL` if you don't want this information; this lets the function works faster.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `Movies.h`

Declared In

`Movies.h`

GetMediaSample2

Retrieves sample data from a media file.

```
OSErr GetMediaSample2 (
 Media                     theMedia,
 UInt8                     *dataOut,
 ByteCount                 maxDataSize,
 ByteCount                 *size,
 TimeValue64               decodeTime,
 TimeValue64               *sampleDecodeTime,
 TimeValue64               *decodeDurationPerSample,
 TimeValue64               *displayOffset,
 SampleDescriptionHandle   sampleDescriptionH,
 ItemCount                 *sampleDescriptionIndex,
 ItemCount                 maxNumberOfSamples,
 ItemCount                 *numberOfSamples,
 MediaSampleFlags          *sampleFlags );
```

Parameters

theMedia

The media for this operation. You obtain this media identifier from such functions as `NewTrackMedia` and `GetTrackMedia`.

dataOut

A pointer to a buffer to receive sample data. The buffer must be large enough to contain at least *maxDataSize* bytes. If you do not want to receive sample data, pass `NULL`.

maxDataSize

The maximum number of bytes allocated to hold the sample data.

size

A pointer to memory where the function returns the number of bytes of sample data returned in the memory area specified by *dataOut*. Set this parameter to `NULL` if you are not interested in this information.

decodeTime

The starting time of the sample to be retrieved in decode time. You must specify this value in the media's time scale.

sampleDecodeTime

A pointer to a time value in decode time. The function updates this time value to indicate the actual time of the returned sample data. (The returned time may differ from the time you specified with the *time* parameter. This will occur if the time you specified falls in the middle of a sample.) If you are not interested in this information, set this parameter to `NULL`.

decodeDurationPerSample

A pointer to a time value in decode time. The Movie Toolbox returns the duration of each sample in the media. Set this parameter to `NULL` if you don't want this information.

displayOffset

A pointer to a time value. The function updates this time value to indicate the display offset of the returned sample. This time value is expressed in the media's time scale. Set this parameter to `NULL` if you don't want this information.

sampleDescriptionH

A handle to a `SampleDescription` structure. The function returns the sample description corresponding to the returned sample data. The function resizes this handle as appropriate. If you don't want a `SampleDescription` structure, set this parameter to `NIL`.

sampleDescriptionIndex

A pointer to a long integer. The function returns an index value to the `SampleDescription` structure that corresponds to the returned sample data. You can retrieve the structure by calling `GetMediaSampleDescription` and passing this index in the *descH* parameter. If you don't want this information, set this parameter to `NIL`.

maxNumberOfSamples

The maximum number of samples to be returned. The Movie Toolbox does not return more samples than you specify with this parameter. If you set this parameter to 0, the Movie Toolbox uses a value that is appropriate for the media, and returns that value in the field referenced by the *numberOfSamples* parameter.

numberOfSamples

A pointer to a long integer. The function updates the field referred to by this parameter with the number of samples it actually returns. If you don't want this information, set this parameter to `NULL`.

sampleFlags

A pointer to a short integer in which the function returns flags that describe the sample. Unused flags are set to 0. If you don't want this information, set this parameter to `NULL`:

`mediaSampleNotSync`

This flag is set to 1 if the sample is not a sync sample and to 0 if the sample is a sync sample.

Return Value

You can access this function's error returns through `GetMoviesError` and `GetMoviesStickyError`. It returns `paramErr` if there is a bad parameter value, `maxSizeToGrowTooSmall` if the sample data is larger than *maxDataSize*, or `noErr` if there is no error.

Discussion

Whereas `GetMediaSample` takes a resizable Handle and a *maxSizeToGrow* parameter, `GetMediaSample2` takes a pointer and a *maxDataSize* parameter. If you want to read a sample into a Handle, you can use the following code:

```
OSErr GetMediaSampleUsingHandle (Media theMedia, Handle dataHOut,
                ByteCount maxSizeToGrow, ByteCount *size,
                TimeValue64 decodeTime, TimeValue64 *sampleDecodeTime,
                TimeValue64 *decodeDurationPerSample,
                TimeValue64 *displayOffset,
                SampleDescriptionHandle sampleDescriptionH,
                ItemCount *sampleDescriptionIndex,
                ItemCount maxNumberOfSamples,
                ItemCount *numberOfSamples,
                MediaSampleFlags *sampleFlags)
 {
     OSErr err = noErr;
     ByteCount actualSize = 0;

     err = GetMediaSample2(theMedia,
                           *dataHOut,
                           GetHandleSize(dataHOut),
                           &actualSize,
                           decodeTime,
                           sampleDecodeTime,
                           decodeDurationPerSample,
                           displayOffset,
                           sampleDescriptionH,
                           sampleDescriptionIndex,
                           maxNumberOfSamples,
                           numberOfSamples,
                           sampleFlags);

     if ((maxSizeToGrowTooSmall == err)
     && ((0 == maxSizeToGrow) || (actualSize <= maxSizeToGrow)) {
         SetHandleSize(dataHOut, actualSize);
         err = MemError();
         if (err) goto bail;

         err = GetMediaSample2(theMedia,
                               *dataHOut,
                               GetHandleSize(dataHOut),
                               &actualSize,
                               decodeTime,
                               sampleDecodeTime,
                               decodeDurationPerSample,
                               displayOffset,
                               sampleDescriptionH,
                               sampleDescriptionIndex,
                               maxNumberOfSamples,
                               numberOfSamples,
                               sampleFlags);
     }

     if( size )
         *size = actualSize;

 bail:
     return err;
 }
```

Version Notes

Introduced in QuickTime 7. This function extends and supersedes `GetMediaSample`. It will only return multiple samples that all have the same decode duration per sample, the same display offset, the same sample description, and the same size per sample.

Availability

Carbon status: Supported C interface file: `Movies.h`

Declared In

`Movies.h`

GetMovieAudioBalance

Returns the balance value for the audio mix of a movie currently playing.

```
OSStatus GetMovieAudioBalance (
 Movie      m,
 Float32    *leftRight,
 UInt32     flags );
```

Parameters

m

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromProperties`, `NewMovieFromFile`, and `NewMovieFromHandle`.

leftRight

On return, a pointer to the current balance setting for the movie. The balance setting is a 32-bit floating-point value that controls the relative volume of the left and right sound channels. A value of 0 sets the balance to neutral. Positive values up to 1.0 shift the balance to the right channel, negative values up to –1.0 to the left channel.

flags

Not used; set to 0.

Return Value

An error code. Returns `noErr` if there is no error.

Discussion

The movie's balance setting is not stored in the movie; it is used only until the movie is closed. See `SetMovieAudioBalance` (page 244).

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `Movies.h`

Declared In

`Movies.h`

GetMovieAudioContext

Returns the current audio context for a movie.

```
OSStatus GetMovieAudioContext (
 Movie                movie,
 QTAudioContextRef    *audioContext);
```

Parameters

movie

The movie.

audioContext

A pointer to a variable to receive the audio context.

Return Value

An error code. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `Movies.h`

Declared In

`Movies.h`

GetMovieAudioFrequencyLevels

Returns the current frequency meter levels of a movie mix.

```
OSStatus GetMovieAudioFrequencyLevels (
 Movie                      m,
 FourCharCode               whatMixToMeter,
 QTAudioFrequencyLevels     *pAveragePowerLevels );
```

Parameters

m

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromProperties`, `NewMovieFromFile`, and `NewMovieFromHandle`.

whatMixToMeter

The applicable mix of audio channels in the movie; see "`Movie Audio Mixes`" (page 268).

pAveragePowerLevels

A pointer to a `QTAudioFrequencyLevels` structure `(page 339)` (page 259).

Return Value

An error code. Returns `noErr` if there is no error.

Discussion

In the structure pointed to by *pAveragePowerLevels*, the *numChannels* field must be set to the number of channels in the movie mix being metered and the *numBands* field must be set to the number of bands being metered (as previously configured). Enough memory for the structure must be allocated to hold 32-bit values for all bands in all channels. This function returns the current frequency meter levels in the *level* field of the structure, with all the band levels for the first channel first, all the band levels for the second channel next and so on.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `Movies.h`

Declared In

`Movies.h`

GetMovieAudioFrequencyMeteringBandFrequencies

Returns the chosen middle frequency for each band in the configured frequency metering of a particular movie mix.

```
OSStatus GetMovieAudioFrequencyMeteringBandFrequencies (
 Movie           m,
 FourCharCode    whatMixToMeter,
 UInt32          numBands,
 Float32         *outBandFrequencies );
```

Parameters

m

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromProperties`, `NewMovieFromFile`, and `NewMovieFromHandle`.

whatMixToMeter

The applicable mix of audio channels in the movie; see "`Movie Audio Mixes`" (page 268).

numBands

The number of bands to examine.

outBandFrequencies

A pointer to an array of frequencies, each expressed in Hz.

Return Value

An error code. Returns `noErr` if there is no error.

Discussion

You can use this function to label a visual meter in a user interface.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `Movies.h`

Declared In

`Movies.h`

GetMovieAudioFrequencyMeteringNumBands

Returns the number of frequency bands being metered for a movie's specified audio mix.

```
OSStatus GetMovieAudioFrequencyMeteringNumBands (
 Movie           m,
 FourCharCode    whatMixToMeter,
 UInt32          *outNumBands );
```

Parameters

m

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromProperties`, `NewMovieFromFile`, and `NewMovieFromHandle`.

whatMixToMeter

The applicable mix of audio channels in the movie; see "`Movie Audio Mixes`" (page 268).

outNumBands

A pointer to memory that stores the number of frequency bands currently being metered for the movie's specified audio mix.

Return Value

An error code. Returns `noErr` if there is no error.

Discussion

See `SetMovieAudioFrequencyMeteringNumBands` (page 246).

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `Movies.h`

Declared In

`Movies.h`

GetMovieAudioGain

Returns the gain value for the audio mix of a movie currently playing.

```
OSStatus GetMovieAudioGain (
 Movie      m,
 Float32    *gain,
 UInt32     flags );
```

Parameters

m

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromProperties`, `NewMovieFromFile`, and `NewMovieFromHandle`.

gain

A 32-bit floating-point gain value of 0 or greater. 0.0 is silent, 0.5 is –6 dB, 1.0 is 0 dB (the audio from the movie is not modified), 2.0 is +6 dB, etc. The gain level can be set higher than 1.0 to allow quiet movies to be boosted in volume. Gain settings higher than 1.0 may result in audio clipping.

flags

Not used; set to 0.

Return Value

An error code. Returns `noErr` if there is no error.

Discussion

The movie gain setting is not stored in the movie; it is used only until the movie is closed. See `SetMovieAudioGain` (page 246).

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `Movies.h`

Declared In

`Movies.h`

GetMovieAudioMute

Returns the mute value for the audio mix of a movie currently playing.

```
OSStatus GetMovieAudioMute (
 Movie      m,
 Boolean    *muted,
 UInt32     flags );
```

Parameters

m

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromProperties`, `NewMovieFromFile`, and `NewMovieFromHandle`.

muted

Returns `TRUE` if the movie audio is currently muted, `FALSE` otherwise.

flags

Not used; set to 0.

Return Value
An error code. Returns `noErr` if there is no error.

Discussion
The movie mute setting is not stored in the movie; it is used only until the movie is closed. See `SetMovieAudioMute` (page 247).

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

GetMovieAudioVolumeLevels

Returns the current volume meter levels of a movie.

```
OSStatus GetMovieAudioVolumeLevels (
 Movie                 m,
 FourCharCode          whatMixToMeter,
 QTAudioVolumeLevels   *pAveragePowerLevels,
 QTAudioVolumeLevels   *pPeakHoldLevels );
```

Parameters

m

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromProperties`, `NewMovieFromFile`, and `NewMovieFromHandle`.

whatMixToMeter

The applicable mix of audio channels in the movie; see “`Movie Audio Mixes`” (page 268).

pAveragePowerLevels
A pointer to a QTAudioVolumeLevels structure that stores the average power level of each channel in the mix, measured in decibels. 0.0 dB for each channel means full volume, –6.0 dB means half volume, –12.0 dB means quarter volume, and –infinite dB means silence. Pass NULL for this parameter if you are not interested in average power levels.

pPeakHoldLevels
A pointer to a QTAudioVolumeLevels structure that stores the peak hold level of each channel in the mix, measured in decibels. 0.0 dB for each channel means full volume, –6.0 dB means half volume, –12.0 dB means quarter volume, and –infinite dB means silence. Pass NULL for this parameter if you are not interested in peak hold levels.

Return Value
An error code. Returns noErr if there is no error.

Discussion
If either *pAveragePowerLevels* or *pPeakHoldLevels* returns non-NULL, it must have the *numChannels* field in its QTAudioVolumeLevels structure set to the number of channels in the movie mix being metered and the memory allocated for the structure must be large enough to hold levels for all those channels.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: Movies.h

Declared In
Movies.h

GetMovieAudioVolumeMeteringEnabled

Returns the enabled or disabled status of volume metering of a particular audio mix of a movie.

```
OSStatus GetMovieAudioVolumeMeteringEnabled (
 Movie           m,
 FourCharCode    whatMixToMeter,
 Boolean         *enabled );
```

Parameters

m
The movie for this operation. Your application obtains this movie identifier from such functions as NewMovie, NewMovieFromProperties, NewMovieFromFile, and NewMovieFromHandle.

whatMixToMeter
The applicable mix of audio channels in the movie; see "Movie Audio Mixes" (page 268).

enabled
Returns TRUE if audio volume metering is enabled, FALSE if it is disabled.

Return Value
An error code. Returns noErr if there is no error.

Discussion
See SetMovieAudioVolumeMeteringEnabled (page 248).

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

GetMovieVisualBrightness

Returns the brightness adjustment for the movie.

```
OSStatus GetMovieVisualBrightness (
 Movie movie,
 Float32 *brightnessOut,
 UInt32 flags );
```

Parameters

`movie`
: The movie.

`brightnessOut`
: Current brightness adjustment.

`flags`
: Reserved. Pass 0.

Return Value
An error code. Returns `noErr` if there is no error.

Discussion
The brightness adjustment for the movie. The value is a Float32 for which -1.0 means full black, 0.0 means no adjustment, and 1.0 means full white. The setting is not stored in the movie. It is only used until the movie is closed, at which time it is not saved.

Version Notes
Introduced in QuickTime 7

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

GetMovieVisualContext

Returns the current visual context for a movie.

```
OSStatus GetMovieVisualContext (
 Movie      movie,
 QTVisualContextRef    *visualContext;
```

Parameters

`movie`
: The movie.

visualContext

A pointer to a variable to receive the visual context.

Return Value

An error code. Returns `noErr` if there is no error. Returns `memFullErr` if memory cannot be allocated. Returns `kQTVisualContextRequiredErr` if the movie is not using a visual context. Returns `paramErr` if the movie or *visualContextOut* is `NULL`.

Discussion

Returns the QTVisualContext object associated with the movie. You are responsible for retaining and releasing the object as needed (that is, if the returned object has not been retained for you). If the visual context was set to `NULL` (see `SetMovieVisualContext`), `noErr` is returned and *visualContextOut* receives `NULL`.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `Movies.h`

Declared In

`Movies.h`

GetMovieVisualContrast

Returns the contrast adjustment for the movie.

```
OSStatus GetMovieVisualContrast (
 Movie           movie,
 Float32         *contrastOut,
 UInt32          flags );
```

Parameters

movie

The movie.

contrastOut

Current contrast adjustment.

flags

Reserved. Pass 0.

Return Value

An error code. Returns `noErr` if there is no error.

Discussion

The contrast adjustment for the movie. The value is a Float32 percentage (1.0f = 100%), such that 0.0 gives solid grey.

Version Notes

Introduced in QuickTime 7

Availability

Carbon status: Supported C interface file: `Movies.h`

Declared In

`Movies.h`

GetMovieVisualHue

Returns the hue adjustment for the movie.

```
OSStatus GetMovieVisualHue (
 Movie movie,
 Float32 *hueOut,
 UInt32 flags );
```

Parameters

movie

The movie.

hueOut

Current hue adjustment. (Float32)

flags

Reserved. Pass 0. (UInt32)

Return Value

An error code. Returns `noErr` if there is no error.

Discussion

The hue adjustment for the movie. The value is a Float32 between -1.0 and 1.0, with 0.0 meaning no adjustment. This adjustment wraps around, such that -1.0 and 1.0 yield the same result. The setting is not stored in the movie. It is only used until the movie is closed, at which time it is not saved.

Version Notes

Introduced in QuickTime 7

Availability

Carbon status: Supported C interface file: `Movies.h`

Declared In

`Movies.h`

GetMovieVisualSaturation

Returns the color saturation adjustment for the movie.

```
OSStatus GetMovieVisualSaturation (
 Movie movie,
 Float32 *saturationOut,
 UInt32 flags );
```

Parameters

movie

The movie.

saturationOut

Current saturation adjustment.(Float32)

flags

Reserved. Pass 0. (UInt32)

Return Value

An error code. Returns `noErr` if there is no error.

Discussion
The color saturation adjustment for the movie. The value is a Float32 percentage (1.0f = 100%), such that 0.0 gives grayscale. The setting is not stored in the movie. It is only used until the movie is closed, at which time it is not saved.

Version Notes
Introduced in QuickTime 7

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

GetTrackAudioGain

Returns the gain value for the audio mix of a track currently playing.

```
OSStatus GetTrackAudioGain (
 Track      t,
 Float32    *gain,
 UInt32     flags );
```

Parameters

`t`

A track identifier, which your application obtains from such functions as `NewMovieTrack` and `GetMovieTrack`.

`gain`

A 32-bit floating-point gain value of 0 or greater. 0.0 is silent, 0.5 is –6 dB, 1.0 is 0 dB (the audio from the track is not modified), 2.0 is +6 dB, etc. The gain level can be set higher than 1.0 to allow quiet tracks to be boosted in volume. Gain settings higher than 1.0 may result in audio clipping.

`flags`

Not used; set to 0.

Return Value
An error code. Returns `noErr` if there is no error.

Discussion
The track gain setting is not stored in the movie; it is used only until the movie is closed. See `SetTrackAudioGain` (page 252).

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

GetTrackAudioMute

Returns the mute value for the audio mix of a track currently playing.

```
OSStatus GetTrackAudioMute (
 Track      t,
 Boolean    *muted,
 UInt32     flags );
```

Parameters

t

A track identifier, which your application obtains from such functions as `NewMovieTrack` and `GetMovieTrack`.

muted

Returns `TRUE` if the track's audio is currently muted, `FALSE` otherwise.

flags

Not used; set to 0.

Return Value

An error code. Returns `noErr` if there is no error.

Discussion

The track's mute setting is not stored in the movie; it is used only until the movie is closed. See `SetTrackAudioMute` (page 252).

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `Movies.h`

Declared In

`Movies.h`

GetTrackEditRate64

Returns the rate of the track edit of a specified track at an indicated time.

```
Fixed GetTrackEditRate64 (
 Track          theTrack,
 TimeValue64    atTime );
```

Parameters

theTrack

A track identifier, which your application obtains from such functions as `NewMovieTrack` and `GetMovieTrack`.

atTime

A 64-bit time value that indicates the time at which the rate of a track edit (of a track identified in the parameter *theTrack*) is to be determined.

Return Value

The rate of the track edit of the specified track at the specified time.

Discussion

This function is useful if you are stepping through track edits directly in your application or if you are a client of QuickTime's base media handler.

Version Notes
Introduced in QuickTime 7. This function is a 64-bit replacement for `GetTrackEditRate`.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

HIMovieViewChangeAttributes

Changes the views attributes.

```
OSStatus HIMovieViewChangeAttributes (
 HIViewRef inView,
 OptionBits inAttributesToSet,
 OptionBits inAttributesToClear );
```

Parameters

`inView`
The HIMovieView.

`inAttributesToSet`
Attributes to set.

`inAttributesToClear`
Attributes to clear.

Return Value
An error code. Returns `noErr` if there is no error.

Discussion
Setting an attribute takes precedence over clearing the attribute.

Version Notes
Introduced in QuickTime 7

Availability
Carbon status: Supported C interface file: `HIMovieView.h`

Declared In
`HIMovieView.h`

HIMovieViewCreate

Creates an HIMovieView object.

```
OSStatus HIMovieViewCreate (
 Movie inMovie,
 OptionBits inAttributes,
 HIViewRef *outMovieView );
```

Parameters

`inMovie`
Initial movie to view; may be `NULL`.

inAttributes
: Initial HIMovieView attributes.

outMovieView
: Points to variable to receive new HIMovieView.

Return Value
Undocumented.

Discussion
If successful, the created view will have a single retain count.

Version Notes
Introduced in QuickTime 7

Availability
Carbon status: Supported C interface file: `HIMovieView.h`

Declared In
`HIMovieView.h`

HIMovieViewGetAttributes

Returns the view's current attributes.

```
OptionBits HIMovieViewGetAttributes (
 HIViewRef inView );
```

Parameters

inView
: The HIMovieView.

Return Value
Undocumented.

Discussion
The view's current attributes are returned.

Version Notes
Introduced in QuickTime 7

Availability
Carbon status: Supported C interface file: `HIMovieView.h`

Declared In
`HIMovieView.h`

HIMovieViewGetControllerBarSize

Returns the size of the visible movie controller bar.

```
HISize HIMovieViewGetControllerBarSize (
 HIViewRef inView );
```

Parameters

inView

The HIMovieView.

Return Value

Undocumented.

Discussion

The size of the visible movie controller bar is returned.

Version Notes

Introduced in QuickTime 7

Availability

Carbon status: Supported C interface file: `HIMovieView.h`

Declared In

`HIMovieView.h`

HIMovieViewGetMovie

Returns the view's current movie.

```
Movie HIMovieViewGetMovie (
 HIViewRef inView );
```

Parameters

inView

The HIMovieView.

Return Value

Undocumented.

Discussion

The view's current movie is returned.

Version Notes

Introduced in QuickTime 7

Availability

Carbon status: Supported C interface file: `HIMovieView.h`

Declared In

`HIMovieView.h`

HIMovieViewGetMovieController

Returns the view's current movie controller.

```
MovieController HIMovieViewGetMovieController (
 HIViewRef inView );
```

Parameters

inView

The HIMovieView.

Return Value

Undocumented.

Discussion

The view's current movie controller is returned.

Version Notes

Introduced in QuickTime 7

Availability

Carbon status: Supported C interface file: `HIMovieView.h`

Declared In

`HIMovieView.h`

HIMovieViewPause

Pauses the view's current movie.

```
OSStatus HIMovieViewPause (
 HIViewRef movieView );
```

Parameters

movieView

The movie view.

Return Value

An error code. Returns `noErr` if there is no error.

Discussion

This is a convenience routine to pause the view's current movie. If the movie is already paused, this function does nothing.

Version Notes

Introduced in QuickTime 7

Availability

Carbon status: Supported C interface file: `HIMovieView.h`

Declared In

`HIMovieView.h`

HIMovieViewPlay

Plays the view's current movie.

```
OSStatus HIMovieViewPlay (
 HIViewRef movieView );
```

Parameters

movieView

The movie view.

Return Value

An error code. Returns `noErr` if there is no error.

Discussion

This is a convenience routine to play the view's current movie. If the movie is already playing, this function does nothing.

Version Notes

Introduced in QuickTime 7

Availability

Carbon status: Supported C interface file: `HIMovieView.h`

Declared In

`HIMovieView.h`

HIMovieViewSetMovie

Sets the view's current movie.

```
OSStatus HIMovieViewSetMovie (
 HIViewRef inView,
 Movie inMovie );
```

Parameters

inView

The HIMovieView.

inMovie

The new movie to display.

Return Value

An error code. Returns `noErr` if there is no error.

Discussion

This routine sets the view's current movie.

Version Notes

Introduced in QuickTime 7

Availability

Carbon status: Supported C interface file: `HIMovieView.h`

Declared In

`HIMovieView.h`

ICMCompressionFrameOptionsCreate

Creates a frame compression options object.

```
OSStatus ICMCompressionFrameOptionsCreate (
 CFAllocatorRef                 allocator,
 ICMCompressionSessionRef       session,
 ICMCompressionFrameOptionsRef  *options );
```

Parameters

allocator

An allocator. Pass `NULL` to use the default allocator.

session

A compression session reference. This reference is returned by `ICMCompressionSessionCreate` (page 106).

options

On return, a reference to a new frame compression options object.

Return Value

An error code. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMCompressionFrameOptionsCreateCopy

Copies a frame compression options object.

```
OSStatus ICMCompressionFrameOptionsCreateCopy (
 CFAllocatorRef                 allocator,
 ICMCompressionFrameOptionsRef  originalOptions,
 ICMCompressionFrameOptionsRef  *copiedOptions );
```

Parameters

allocator

An allocator. Pass `NULL` to use the default allocator.

originalOptions

A frame compression options reference. This reference is returned by `ICMCompressionFrameOptionsCreate` (page 96).

copiedOptions

On return, a reference to a copy of the frame compression options object passed in *originalOptions*.

Return Value

An error code. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMCompressionFrameOptionsGetForceKeyFrame

Retrieves the force key frame flag.

```
Boolean ICMCompressionFrameOptionsGetForceKeyFrame (
 ICMCompressionFrameOptionsRef   options );
```

Parameters

`options`
A compression frame options reference. This reference is returned by `ICMCompressionFrameOptionsCreate` (page 96).

Return Value
Returns `TRUE` if frames are forced to be compressed as key frames, `FALSE` otherwise.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMCompressionFrameOptionsGetFrameType

Retrieves the `frame` type setting.

```
ICMFrameType ICMCompressionFrameOptionsGetFrameType (
 ICMCompressionFrameOptionsRef   options );
```

Parameters

`options`
A compression frame options reference. This reference is returned by `ICMCompressionFrameOptionsCreate` (page 96).

Return Value
On return, one of the `frame` types listed below.

Discussion
This function can return one of these constants:

`kICMFrameType_I = 'I'`
An I frame.

`kICMFrameType_P = 'P'`
A P frame.

`kICMFrameType_B = 'B'`

A B frame.

`kICMFrameType_Unknown = 0`

A frame of unknown type.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMCompressionFrameOptionsGetProperty

Retrieves the value of a specific property of a compression frame options object.

```
OSStatus ICMCompressionFrameOptionsGetProperty (
 ICMCompressionFrameOptionsRef   options,
 ComponentPropertyClass          inPropClass,
 ComponentPropertyID             inPropID,
 ByteCount                       inPropValueSize,
 ComponentValuePtr               outPropValueAddress,
 ByteCount                       *outPropValueSizeUsed );
```

Parameters

`options`

A compression frame options reference. This reference is returned by `ICMCompressionFrameOptionsCreate` (page 96).

`inPropClass`

Pass the following constant to define the property class:

`kComponentPropertyClassPropertyInfo = 'pnfo'`

The property information class.

inPropID

Pass one of these constants to define the property ID:

`kComponentPropertyInfoList = 'list'`

An array of `CFData` values, one for each property.

`kComponentPropertyCacheSeed = 'seed'`

A property cache seed value.

`kComponentPropertyCacheFlags = 'flgs'`

One of the `kComponentPropertyCache` flags:

`kComponentPropertyCacheFlagNotPersistent`

Property metadata should not be saved in persistent cache.

`kComponentPropertyCacheFlagIsDynamic`

Property metadata should not cached at all.

`kComponentPropertyExtendedInfo = 'meta'`

A `CFDictionary` with extended property information.

outPropType

A pointer to the type of the returned property's value.

outPropValueAddress

A pointer to a variable to receive the returned property's value.

outPropValueSizeUsed

On return, a pointer to the number of bytes actually used to store the property.

Return Value

An error code. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMCompressionFrameOptionsGetPropertyInfo

Retrieves information about properties of a compression frame options object.

```
OSStatus ICMCompressionFrameOptionsGetPropertyInfo (
 ICMCompressionFrameOptionsRef   options,
 ComponentPropertyClass          inPropClass,
 ComponentPropertyID             inPropID,
 ComponentValueType              *outPropType,
 ByteCount                       *outPropValueSize,
 UInt32                          *outPropertyFlags );
```

Parameters

options

A compression frame options reference. This reference is returned by `ICMCompressionFrameOptionsCreate` (page 96).

inPropClass

Pass the following constant to define the property class:

`kComponentPropertyClassPropertyInfo = 'pnfo'`

The property information class.

inPropID

Pass one of these constants to define the property ID:

`kComponentPropertyInfoList = 'list'`

An array of `CFData` values, one for each property.

`kComponentPropertyCacheSeed = 'seed'`

A property cache seed value.

`kComponentPropertyCacheFlags = 'flgs'`

One of the `kComponentPropertyCache` flags:

`kComponentPropertyCacheFlagNotPersistent`

Property metadata should not be saved in persistent cache.

`kComponentPropertyCacheFlagIsDynamic`

Property metadata should not cached at all.

`kComponentPropertyExtendedInfo = 'meta'`

A `CFDictionary` with extended property information.

outPropType

A pointer to the type of the returned property's value.

outPropValueSize

A pointer to the size of the returned property's value.

outPropFlags

On return, a pointer to flags representing the requested information about the property.

Return Value

An error code. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMCompressionFrameOptionsGetTypeID

Returns the type ID for the current frame compression options object.

```
CFTypeID ICMCompressionFrameOptionsGetTypeID ( void );
```

Return Value
A `CFTypeID` value.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMCompressionFrameOptionsRelease

Decrements the retain count of a frame compression options object.

```
void ICMCompressionFrameOptionsRelease (
 ICMCompressionFrameOptionsRef    options );
```

Parameters

`options`
A reference to a frame compression options object.This reference is returned by `ICMCompressionFrameOptionsCreate` (page 96). If you pass `NULL`, nothing happens.

Discussion
If the retain count drops to 0, the object is disposed.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMCompressionFrameOptionsRetain

Increments the retain count of a frame compression options object.

```
ICMCompressionFrameOptionsRef ICMCompressionFrameOptionsRetain (
 ICMCompressionFrameOptionsRef    options );
```

Parameters

options

A reference to a frame compression options object.This reference is returned by `ICMCompressionFrameOptionsCreate` (page 96). If you pass `NULL`, nothing happens.

Return Value

A copy of the object reference passed in *options*, for convenience.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMCompressionFrameOptionsSetForceKeyFrame

Forces frames to be compressed as key frames.

```
OSStatus ICMCompressionFrameOptionsSetForceKeyFrame (
 ICMCompressionFrameOptionsRef   options,
 Boolean                         forceKeyFrame );
```

Parameters

forceKeyFrame

options

A compression frame options reference. This reference is returned by `ICMCompressionFrameOptionsCreate` (page 96).

forceKeyFrame

Pass `TRUE` to force frames to be compressed as key frames, `FALSE` otherwise.

Return Value

An error code. Returns `noErr` if there is no error.

Discussion

The compressor must obey this flag if set. By default it is set `FALSE`.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMCompressionFrameOptionsSetFrameType

Requests a frame be compressed as a particular frame type.

```
OSStatus ICMCompressionFrameOptionsSetFrameType (
 ICMCompressionFrameOptionsRef   options,
 ICMFrameType                    frameType );
```

Parameters

options

A compression frame options reference. This reference is returned by `ICMCompressionFrameOptionsCreate` (page 96).

frameType

A constant that identifies a frame type. Pass one of the following but do not assume that there are no other frame types:

`kICMFrameType_I = 'I'`

An I frame.

`kICMFrameType_P = 'P'`

A P frame.

`kICMFrameType_B = 'B'`

A B frame.

`kICMFrameType_Unknown = 0`

A frame of unknown type.

Return Value

An error code. Returns `noErr` if there is no error.

Discussion

The frame type setting may be ignored by the compressor if it is not appropriate. By default it is set to `kICMFrameType_Unknown`.

Do not assume that `kICMFrameType_I` sets a key frame; if you need a key frame, call `ICMCompressionFrameOptionsSetForceKeyFrame` (page 102).

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMCompressionFrameOptionsSetProperty

Sets the value of a specific property of a compression frame options object.

```
OSStatus ICMCompressionFrameOptionsSetProperty (
 ICMCompressionFrameOptionsRef   options,
 ComponentPropertyClass          inPropClass,
 ComponentPropertyID             inPropID,
 ByteCount                       inPropValueSize,
 ConstComponentValuePtr          inPropValueAddress );
```

Parameters

options

A compression frame options reference. This reference is returned by `ICMCompressionFrameOptionsCreate` (page 96).

inPropClass

Pass the following constant to define the property class:

`kComponentPropertyClassPropertyInfo = 'pnfo'`

The property information class.

inPropID

Pass one of these constants to define the property ID:

`kComponentPropertyInfoList = 'list'`

An array of `CFData` values, one for each property.

`kComponentPropertyCacheSeed = 'seed'`

A property cache seed value.

`kComponentPropertyCacheFlags = 'flgs'`

One of the `kComponentPropertyCache` flags:

`kComponentPropertyCacheFlagNotPersistent`

Property metadata should not be saved in persistent cache.

`kComponentPropertyCacheFlagIsDynamic`

Property metadata should not cached at all.

`kComponentPropertyExtendedInfo = 'meta'`

A `CFDictionary` with extended property information.

inPropValueSize

The size of the property value to be set.

inPropValueAddress

A pointer to the value of the property to be set.

Return Value

An error code. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMCompressionSessionBeginPass

Announces the start of a specific compression pass.

```
OSStatus ICMCompressionSessionBeginPass (
 ICMCompressionSessionRef      session,
 ICMCompressionPassModeFlags   passModeFlags,
 UInt32                        flags );
```

Parameters

session

A compression session reference. This reference is returned by `ICMCompressionSessionCreate` (page 106).

passModeFlags

Flags that describe how the compressor should behave in this pass of multipass encoding:

`kICMCompressionPassMode_OutputEncodedFrames = 1L<<0`

Output encoded frames.

`kICMCompressionPassMode_NoSourceFrames = 1L<<1`

The client need not provide source frame buffers.

`kICMCompressionPassMode_WriteToMultiPassStorage = 1L<<2`

The compressor may write private data to multipass storage.

`kICMCompressionPassMode_ReadFromMultiPassStorage = 1L<<3`

The compressor may read private data from multipass storage.

flags

Reserved. Set to 0.

Return Value

An error code. Returns `noErr` if there is no error.

Discussion

The source frames and frame options for each display time should be the same across passes. During multipass compression, valid *displayTimeStamp* values must be passed to `ICMCompressionSessionEncodeFrame`, because they are used to index the compressor's stored state.

During an analysis pass (`kICMCompressionPassMode_WriteToMultiPassStorage`), the compressor does not output encoded frames but records compressor-private information for each frame. During repeated analysis passes and the encoding pass (`kICMCompressionPassMode_ReadFromMultiPassStorage`), the compressor may refer to this information for other frames and use it to improve encoding. During an encoding pass (`kICMCompressionPassMode_OutputEncodedFrames`), the compressor must output encoded frames. If the compressor sets the `kICMCompressionPassMode_NoSourceFrames` flag for the pass, the client may pass `NULL` pixel buffers to `ICMCompressionSessionEncodeFrame`.

By default, the ICM provides local storage that lasts only until the compression session is disposed. If the client provides custom multipass storage, passes may be performed at different times or on different machines; segments of each pass may even be distributed.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMCompressionSessionCompleteFrames

Forces a compression session to complete encoding frames.

```
OSStatus ICMCompressionSessionCompleteFrames (
 ICMCompressionSessionRef    session,
 Boolean                     completeAllFrames,
 TimeValue64                 completeUntilDisplayTimeStamp,
 TimeValue64                 nextDisplayTimeStamp );
```

Parameters

session
: A reference to a video compression session, returned by a previous call to `ICMCompressionSessionCreate` (page 106).

completeAllFrames
: Pass `TRUE` to direct the session to complete all pending frames.

completeUntilDisplayTimeStamp
: A 64-bit time value that represents the display time up to which to complete frames. This value is ignored if *completeAllFrames* is `TRUE`.

nextDisplayTimeStamp
: A 64-bit time value that represents the display time of the next frame that should be passed to `EncodeFrame`. This value is ignored unless `ICMCompressionSessionOptionsSetDurationsNeeded` set `TRUE` and `kICMValidTime_DisplayDurationIsValid` was 0 in *validTimeFlags* in the last call to `ICMCompressionSessionEncodeFrame`.

Return Value

Returns an error code, or 0 if there is no error. The function may return before frames are completed if the encoded frame callback routine returns an error.

Discussion

Call this function to force a compression session to complete encoding frames. Set *completeAllFrames* to direct the session to complete all pending frames. If *completeAllFrames* is false, only frames with display time stamps up to and including the time passed in *completeUntilDisplayTimeStamp* will be encoded. If `ICMCompressionSessionOptionsSetDurationsNeeded` set `TRUE` and you are passing valid display timestamps but not display durations to `ICMCompressionSessionEncodeFrame`, pass in *nextDisplayTimeStamp* the display timestamp of the next frame that would be passed to `EncodeFrame`.

Version Notes

Introduced in QuickTime 7

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMCompressionSessionCreate

Creates a compression session for a specified codec type.

```
OSStatus ICMCompressionSessionCreate (
 CFAllocatorRef                     allocator,
 int                                width,
 int                                height,
 CodecType                          cType,
 TimeScale                          timescale,
 ICMCompressionSessionOptionsRef    compressionOptions,
 CFDictionaryRef                    sourcePixelBufferAttributes,
 ICMEncodedFrameOutputRecord        *encodedFrameOutputRecord,
 ICMCompressionSessionRef           *compressionSessionOut );
```

Parameters

allocator

An allocator for the session. Pass `NULL` to use the default allocator.

width

The width of frames. Pass 0 to let the compressor control the width.

height

The height of frames. Pass 0 to let the compressor control the height.

cType

The codec type.

timescale

The timescale to be used for all time stamps and durations used in the session.

compressionOptions

A reference to a settings object that configures the session. You create such an object by calling `ICMCompressionSessionOptionsCreate`. You can then use these constants to set its properties:

`kICMUnlimitedFrameDelayCount`

No limit on the number of frames in the compression window.

`kICMUnlimitedFrameDelayTime`

No time limit on the frames in the compression window.

`kICMUnlimitedCPUTimeBudget`

No CPU time limit on compression.

sourcePixelBufferAttributes

Required attributes for source pixel buffers, used when creating a pixel buffer pool for source frames. If you do not want the ICM to create one for you, pass `NULL`. Using pixel buffers not allocated by the ICM may increase the chance that it will be necessary to copy image data.

encodedFrameOutputRecord

The callback that will receive encoded frames.

compressionSessionOut

Points to a variable to receive the created session object.

Return Value

An error code. Returns `noErr` if there is no error.

Discussion

Some compressors do not support arbitrary source dimensions, and may override the suggested width and height.

Version Notes

Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMCompressionSessionEncodeFrame

Presents video frames to a compression session.

```
OSStatus ICMCompressionSessionEncodeFrame (
 ICMCompressionSessionRef           session,
 CVPixelBufferRef                   pixelBuffer,
 TimeValue64                        displayTimeStamp,
 TimeValue64                        displayDuration,
 ICMValidTimeFlags                  validTimeFlags,
 ICMCompressionFrameOptionsRef      frameOptions,
 ICMSourceTrackingCallbackRecord    *sourceTrackingCallback,
 void                               *sourceFrameRefCon );
```

Parameters

session
A reference to a video compression session, returned by a previous call to `ICMCompressionSessionCreate` (page 106).

pixelBuffer
A reference to a buffer containing a source image to be compressed, which must have a nonzero reference count. The session will retain it as long as necessary. The client should not modify the pixel buffer's pixels until the pixel buffer release callback is called. In a multipass encoding session pass, where the compressor suggested the flag `kICMCompressionPassMode_NoSourceFrames`, you may pass `NULL` in this parameter.

displayTimeStamp
A 64-bit time value that represents the display time of the frame, using the time scale passed to `ICMCompressionSessionCreate`. If you pass a valid value, set the `kICMValidTime_DisplayTimeStampIsValid` flag in the *validTimeFlags* parameter (below).

displayDuration
A 64-bit time value that represents the display duration of the frame, using the time scale passed to `ICMCompressionSessionCreate`. If you pass a valid value, set the `kICMValidTime_DisplayDurationIsValid` flag in the *validTimeFlags* parameter (below).

validTimeFlags
Flags to indicate which of the values passed in *displayTimeStamp* and *displayDuration* are valid:

`kICMValidTime_DisplayTimeStampIsValid`
The time value passed in *displayTimeStamp* is valid.

`kICMValidTime_DisplayDurationIsValid`
The time value passed in *displayDuration* is valid.

frameOptions
Options for this frame. Currently not used; pass `NULL`.

sourceTrackingCallback
A pointer to a callback to be notified about the status of this source frame. Pass `NULL` if you do not require notification.

sourceFrameRefCon
A reference constant to be passed to your callback. Use this parameter to point to a data structure containing any information your callback needs.

Return Value
Returns an error code, or 0 if there is no error. Encoded frames may or may not be output before the function returns.

Discussion
The session will retain the pixel buffer as long as necessary, and the client should not modify the pixel data until the session releases it. The most practical way to deal with this is by allocating pixel buffers from a pool. The client may fill in both, either, or neither of *displayTimeStamp* and *displayDuration*, but should set the appropriate flags to indicate which are valid. If the client needs to track the progress of a source frame, it should provide a source tracking callback. If multipass compression is enabled, calls to this function must be bracketed by calls to `ICMCompressionSessionBeginPass` and `ICMCompressionSessionEndPass`.

Version Notes
Introduced in QuickTime 7

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMCompressionSessionEndPass

Announces the end of a pass.

```
OSStatus ICMCompressionSessionEndPass (
 ICMCompressionSessionRef   session );
```

Parameters

session
A compression session reference. This reference is returned by `ICMCompressionSessionCreate` (page 106).

Return Value
An error code. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMCompressionSessionGetImageDescription

Retrieves the image description for a video compression session.

```
OSStatus ICMCompressionSessionGetImageDescription (
 ICMCompressionSessionRef    session,
 ImageDescriptionHandle      *imageDescOut );
```

Parameters

session

A reference to a video compression session, returned by a previous call to `ICMCompressionSessionCreate` (page 106).

imageDescOut

A handle to an `ImageDescription` structure. The caller must *not* dispose of this handle; the ICM will dispose of it when the compression session is disposed.

Return Value

Returns an error code, or 0 if there is no error. For some codecs, this function may fail if called before the first frame is compressed.

Discussion

Multiple calls to this function return the same handle.

Version Notes

Introduced in QuickTime 7

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMCompressionSessionGetPixelBufferPool

Returns a pool that can provide ideal source pixel buffers for a compression session.

```
CVPixelBufferPoolRef ICMCompressionSessionGetPixelBufferPool (
 ICMCompressionSessionRef   session );
```

Parameters

session

A compression session reference. This reference is returned by `ICMCompressionSessionCreate` (page 106).

Return Value

A reference to a pool of pixel buffers. The compression session creates this pixel buffer pool based on the compressor's pixel buffer attributes and any pixel buffer attributes passed to `ICMCompressionSessionCreate` (page 106).

Discussion

A new compression session builds this pixel buffer pool based on the compressor's pixel buffer attributes and any pixel buffer attributes passed in to `ICMCompressionSessionCreate`. If the source pixel buffer attributes and the compressor pixel buffer attributes cannot be reconciled, the pool is based on the source pixel buffer attributes and the ICM converts each pixel buffer internally.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMCompressionSessionGetProperty

Retrieves the value of a specific property of a compression session.

```
OSStatus ICMCompressionSessionGetProperty (
 ICMCompressionSessionRef   session,
 ComponentPropertyClass     inPropClass,
 ComponentPropertyID        inPropID,
 ByteCount                  inPropValueSize,
 ComponentValuePtr          outPropValueAddress,
 ByteCount                  *outPropValueSizeUsed );
```

Parameters

`session`

A compression session reference. This reference is returned by `ICMCompressionSessionCreate` (page 106).

`inPropClass`

Pass the following constant to define the property class:

`kComponentPropertyClassPropertyInfo = 'pnfo'`

The property information class.

`inPropID`

Pass one of these constants to define the property ID:

`kComponentPropertyInfoList = 'list'`

An array of `CFData` values, one for each property.

`kComponentPropertyCacheSeed = 'seed'`

A property cache seed value.

`kComponentPropertyCacheFlags = 'flgs'`

One of the `kComponentPropertyCache` flags:

`kComponentPropertyCacheFlagNotPersistent`

Property metadata should not be saved in persistent cache.

`kComponentPropertyCacheFlagIsDynamic`

Property metadata should not cached at all.

`kComponentPropertyExtendedInfo = 'meta'`

A `CFDictionary` with extended property information.

`outPropType`

A pointer to the type of the returned property's value.

outPropValueAddress

A pointer to a variable to receive the returned property's value.

outPropValueSizeUsed

On return, a pointer to the number of bytes actually used to store the property.

Return Value

An error code. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMCompressionSessionGetPropertyInfo

Retrieves information about properties of a compression session.

```
OSStatus ICMCompressionSessionGetPropertyInfo (
 ICMCompressionSessionRef   session,
 ComponentPropertyClass     inPropClass,
 ComponentPropertyID        inPropID,
 ComponentValueType         *outPropType,
 ByteCount                  *outPropValueSize,
 UInt32                     *outPropertyFlags );
```

Parameters

session

A compression session reference. This reference is returned by `ICMCompressionSessionCreate` (page 106).

inPropClass

Pass the following constant to define the property class:

`kComponentPropertyClassPropertyInfo = 'pnfo'`

The property information class.

`inPropID`

Pass one of these constants to define the property ID:

`kComponentPropertyInfoList = 'list'`

An array of `CFData` values, one for each property.

`kComponentPropertyCacheSeed = 'seed'`

A property cache seed value.

`kComponentPropertyCacheFlags = 'flgs'`

One of the `kComponentPropertyCache` flags:

`kComponentPropertyCacheFlagNotPersistent`

Property metadata should not be saved in persistent cache.

`kComponentPropertyCacheFlagIsDynamic`

Property metadata should not cached at all.

`kComponentPropertyExtendedInfo = 'meta'`

A `CFDictionary` with extended property information.

`outPropType`

A pointer to the type of the returned property's value.

`outPropValueSize`

A pointer to the size of the returned property's value.

`outPropFlags`

On return, a pointer to flags representing the requested information about the property.

Return Value

An error code. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMCompressionSessionGetTimeScale

Retrieves the time scale for a compression session.

```
TimeScale ICMCompressionSessionGetTimeScale (
 ICMCompressionSessionRef   session );
```

Parameters

`session`

A compression session reference. This reference is returned by `ICMCompressionSessionCreate` (page 106).

Return Value

The time scale for the compression session.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMCompressionSessionGetTypeID

Returns the type ID for the current compression session.

```
CFTypeID ICMCompressionSessionGetTypeID ( void );
```

Return Value
A `CFTypeID` value.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMCompressionSessionOptionsCreate

Creates a compression session options object.

```
OSStatus ICMCompressionSessionOptionsCreate (
 CFAllocatorRef                     allocator,
 ICMCompressionSessionOptionsRef   *options );
```

Parameters

`allocator`
An allocator. Pass `NULL` to use the default allocator.

`options`
On return, a reference to a new compression session options object.

Return Value
An error code. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMCompressionSessionOptionsCreateCopy

Copies a compression session options object.

```
OSStatus ICMCompressionSessionOptionsCreateCopy (
 CFAllocatorRef                   allocator,
 ICMCompressionSessionOptionsRef   originalOptions,
 ICMCompressionSessionOptionsRef   *copiedOptions );
```

Parameters

allocator

An allocator. Pass `NULL` to use the default allocator.

originalOptions

A compression session options reference. This reference is returned by `ICMCompressionSessionOptionsCreate` (page 114).

copiedOptions

On return, a reference to a copy of the compression session options object passed in *originalOptions*.

Return Value

An error code. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMCompressionSessionOptionsGetAllowFrameReordering

Retrieves the allow frame reordering flag.

```
Boolean ICMCompressionSessionOptionsGetAllowFrameReordering (
 ICMCompressionSessionOptionsRef   options );
```

Parameters

options

A compression session options reference. This reference is returned by `ICMCompressionSessionOptionsCreate` (page 114).

Return Value

Returns `TRUE` if frame reordering is allowed, `FALSE` otherwise.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMCompressionSessionOptionsGetAllowFrameTimeChanges

Retrieves the allow frame time changes flag.

```
Boolean ICMCompressionSessionOptionsGetAllowFrameTimeChanges (
 ICMCompressionSessionOptionsRef   options );
```

Parameters

options

A compression session options reference. This reference is returned by `ICMCompressionSessionOptionsCreate` (page 114).

Return Value

Returns `TRUE` if the compressor is allowed to modify frame times, `FALSE` otherwise.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMCompressionSessionOptionsGetAllowTemporalCompression

Retrieves the allow temporal compression flag.

```
Boolean ICMCompressionSessionOptionsGetAllowTemporalCompression (
 ICMCompressionSessionOptionsRef   options );
```

Parameters

options

A compression session options reference. This reference is returned by `ICMCompressionSessionOptionsCreate` (page 114).

Return Value

Returns `TRUE` if temporal compression is allowed, `FALSE` otherwise.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMCompressionSessionOptionsGetDurationsNeeded

Retrieves the durations needed flag.

```
Boolean ICMCompressionSessionOptionsGetDurationsNeeded (
 ICMCompressionSessionOptionsRef   options );
```

Parameters

options

A compression session options reference. This reference is returned by `ICMCompressionSessionOptionsCreate` (page 114).

Return Value

Returns `TRUE` if the durations of outputted frames must be calculated, `FALSE` otherwise.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMCompressionSessionOptionsGetMaxKeyFrameInterval

Retrieves the maximum key frame interval.

```
SInt32 ICMCompressionSessionOptionsGetMaxKeyFrameInterval (
 ICMCompressionSessionOptionsRef   options );
```

Parameters

options

A compression session options reference. This reference is returned by `ICMCompressionSessionOptionsCreate` (page 114).

Return Value

Returns the maximum key frame interval.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMCompressionSessionOptionsGetProperty

Retrieves the value of a specific property of a compression session options object.

```
OSStatus ICMCompressionSessionOptionsGetProperty (
 ICMCompressionSessionOptionsRef   options,
 ComponentPropertyClass            inPropClass,
 ComponentPropertyID               inPropID,
 ByteCount                         inPropValueSize,
 ComponentValuePtr                 outPropValueAddress,
 ByteCount                         *outPropValueSizeUsed );
```

Parameters

options

A compression session options reference. This reference is returned by `ICMCompressionSessionOptionsCreate` (page 114).

inPropClass

Pass the following constant to define the property class:

`kComponentPropertyClassPropertyInfo = 'pnfo'`

The property information class.

inPropID

Pass one of these constants to define the property ID:

`kComponentPropertyInfoList = 'list'`

An array of `CFData` values, one for each property.

`kComponentPropertyCacheSeed = 'seed'`

A property cache seed value.

`kComponentPropertyCacheFlags = 'flgs'`

One of the `kComponentPropertyCache` flags:

`kComponentPropertyCacheFlagNotPersistent`

Property metadata should not be saved in persistent cache.

`kComponentPropertyCacheFlagIsDynamic`

Property metadata should not cached at all.

`kComponentPropertyExtendedInfo = 'meta'`

A `CFDictionary` with extended property information.

outPropType

A pointer to the type of the returned property's value.

outPropValueAddress

A pointer to a variable to receive the returned property's value.

outPropValueSizeUsed

On return, a pointer to the number of bytes actually used to store the property.

Return Value

An error code. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMCompressionSessionOptionsGetPropertyInfo

Retrieves information about properties of a compression session options object.

```
OSStatus ICMCompressionSessionOptionsGetPropertyInfo (
 ICMCompressionSessionOptionsRef   options,
 ComponentPropertyClass            inPropClass,
 ComponentPropertyID               inPropID,
 ComponentValueType                *outPropType,
 ByteCount                         *outPropValueSize,
 UInt32                            *outPropertyFlags );
```

Parameters

`options`
A compression session options reference. This reference is returned by `ICMCompressionSessionOptionsCreate` (page 114).

`inPropClass`
Pass the following constant to define the property class:

`kComponentPropertyClassPropertyInfo = 'pnfo'`
The property information class.

`inPropID`
Pass one of these constants to define the property ID:

`kComponentPropertyInfoList = 'list'`
An array of `CFData` values, one for each property.

`kComponentPropertyCacheSeed = 'seed'`
A property cache seed value.

`kComponentPropertyCacheFlags = 'flgs'`
One of the `kComponentPropertyCache` flags:

`kComponentPropertyCacheFlagNotPersistent`
Property metadata should not be saved in persistent cache.

`kComponentPropertyCacheFlagIsDynamic`
Property metadata should not cached at all.

`kComponentPropertyExtendedInfo = 'meta'`
A `CFDictionary` with extended property information.

`outPropType`
A pointer to the type of the returned property's value.

`outPropValueSize`
A pointer to the size of the returned property's value.

`outPropFlags`
On return, a pointer to flags representing the requested information about the property.

Return Value

An error code. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMCompressionSessionOptionsGetTypeID

Returns the type ID for the current compression session options object.

```
CFTypeID ICMCompressionSessionOptionsGetTypeID ( void );
```

Return Value

A `CFTypeID` value.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMCompressionSessionOptionsRelease

Decrements the retain count of a compression session options object.

```
void ICMCompressionSessionOptionsRelease (
 ICMCompressionSessionOptionsRef    options );
```

Parameters

`options`

A reference to a compression session options object. This reference is returned by `ICMCompressionSessionOptionsCreate` (page 114). If you pass `NULL`, nothing happens.

Discussion

If the retain count drops to 0, the object is disposed.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMCompressionSessionOptionsRetain

Increments the retain count of a compression session options object.

```
ICMCompressionSessionOptionsRef ICMCompressionSessionOptionsRetain (
 ICMCompressionSessionOptionsRef    options );
```

Parameters

options

A reference to a compression session options object. This reference is returned by `ICMCompressionSessionOptionsCreate` (page 114). If you pass `NULL`, nothing happens.

Return Value

A copy of the object reference passed in *options*, for convenience.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMCompressionSessionOptionsSetAllowFrameReordering

Enables frame reordering.

```
OSStatus ICMCompressionSessionOptionsSetAllowFrameReordering (
 ICMCompressionSessionOptionsRef   options,
 Boolean                           allowFrameReordering );
```

Parameters

options

A compression session options reference. This reference is returned by `ICMCompressionSessionOptionsCreate` (page 114).

allowFrameReordering

Pass `TRUE` to enable frame reordering, `FALSE` to disable it.

Return Value

An error code. Returns `noErr` if there is no error.

Discussion

To encode B-frames a compressor must reorder frames, which means that the order in which they will be emitted and stored (the decode order) is different from the order in which they were presented to the compressor (the display order). By default, frame reordering is disabled. To encode using B-frames, you must call this function, passing `TRUE`.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMCompressionSessionOptionsSetAllowFrameTimeChanges

Allows the compressor to modify frame times.

```
OSStatus ICMCompressionSessionOptionsSetAllowFrameTimeChanges (
 ICMCompressionSessionOptionsRef   options,
 Boolean                           allowFrameTimeChanges );
```

Parameters

options

A compression session options reference. This reference is returned by `ICMCompressionSessionOptionsCreate` (page 114).

allowFrameTimeChanges

Pass `TRUE` to let the compressor to modify frame times, `FALSE` to prohibit it.

Return Value

An error code. Returns `noErr` if there is no error.

Discussion

Some compressors are able to identify and coalesce runs of identical frames and output single frames with longer durations, or output frames at a different frame rate from the original. This feature is controlled by the allow frame time changes flag. By default, this flag is set to false, which forces compressors to emit one encoded frame for every source frame and preserve frame display times.

This function replaces the practice of having compressors return special high similarity values to indicate that frames could be dropped.

If you want to let the compressor modify frame times in order to improve compression performance, you should allow frame time changes.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMCompressionSessionOptionsSetAllowTemporalCompression

Enables temporal compression.

```
OSStatus ICMCompressionSessionOptionsSetAllowTemporalCompression (
 ICMCompressionSessionOptionsRef   options,
 Boolean                           allowTemporalCompression );
```

Parameters

options

A compression session options reference. This reference is returned by `ICMCompressionSessionOptionsCreate` (page 114).

allowTemporalCompression

Pass `TRUE` to enable temporal compression, `FALSE` to disable it.

Return Value
An error code. Returns `noErr` if there is no error.

Discussion
By default, temporal compression is disabled. If you want temporal compression for P-frames or B-frames you must call this function and pass `TRUE`.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMCompressionSessionOptionsSetDurationsNeeded

Indicates that the durations of outputted frames must be calculated.

```
OSStatus ICMCompressionSessionOptionsSetDurationsNeeded (
 ICMCompressionSessionOptionsRef   options,
 Boolean                           decodeDurationsNeeded );
```

Parameters

options

A compression session options reference. This reference is returned by `ICMCompressionSessionOptionsCreate` (page 114).

decodeDurationsNeeded

Pass `TRUE` to indicate that durations must be calculated, `FALSE` otherwise.

Return Value
An error code. Returns `noErr` if there is no error.

Discussion
If this flag is set and source frames are provided with times but not durations, then frames will be delayed so that durations can be calculated as the difference between one frame's time stamp and the next frame's time stamp. By default this flag is 0, so frames will not be delayed in order to calculate durations.

If you are passing encoded frames to `AddMediaSampleFromEncodedFrame`, you must call this function and pass `TRUE`.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMCompressionSessionOptionsSetMaxKeyFrameInterval

Sets the maximum interval between key frames.

```
OSStatus ICMCompressionSessionOptionsSetMaxKeyFrameInterval (
 ICMCompressionSessionOptionsRef   options,
 SInt32                            maxKeyFrameInterval );
```

Parameters

options

A compression session options reference. This reference is returned by `ICMCompressionSessionOptionsCreate` (page 114).

maxKeyFrameInterval

The maximum interval between key frames, also known as the key frame rate.

Return Value

An error code. Returns `noErr` if there is no error.

Discussion

Compressors are allowed to generate key frames more frequently if this would result in more efficient compression. The default key frame interval is 0, which indicates that the compressor should choose where to place all key frames.

This is a break with previous practice, which used a key frame rate of 0 to disable temporal compression.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMCompressionSessionOptionsSetProperty

Sets the value of a specific property of a compression session options object.

```
OSStatus ICMCompressionSessionOptionsSetProperty (
 ICMCompressionSessionOptionsRef   options,
 ComponentPropertyClass            inPropClass,
 ComponentPropertyID               inPropID,
 ByteCount                         inPropValueSize,
 ConstComponentValuePtr            inPropValueAddress );
```

Parameters

options

A compression session options reference. This reference is returned by `ICMCompressionSessionOptionsCreate` (page 114).

inPropClass

Pass the following constant to define the property class:

`kComponentPropertyClassPropertyInfo = 'pnfo'`

The property information class.

inPropID

Pass one of these constants to define the property ID:

`kComponentPropertyInfoList = 'list'`

An array of `CFData` values, one for each property.

`kComponentPropertyCacheSeed = 'seed'`

A property cache seed value.

`kComponentPropertyCacheFlags = 'flgs'`

One of the `kComponentPropertyCache` flags:

`kComponentPropertyCacheFlagNotPersistent`

Property metadata should not be saved in persistent cache.

`kComponentPropertyCacheFlagIsDynamic`

Property metadata should not cached at all.

`kComponentPropertyExtendedInfo = 'meta'`

A `CFDictionary` with extended property information.

inPropValueSize

The size of the property value to be set.

inPropValueAddress

A pointer to the value of the property to be set.

Return Value

An error code. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMCompressionSessionProcessBetweenPasses

Lets the compressor perform processing between passes.

```
OSStatus ICMCompressionSessionProcessBetweenPasses (
 ICMCompressionSessionRef      session,
 UInt32                        flags,
 Boolean                       *interpassProcessingDoneOut,
 ICMCompressionPassModeFlags   *requestedNextPassModeFlagsOut );
```

Parameters

session

A compression session reference. This reference is returned by `ICMCompressionSessionCreate` (page 106).

flags

Reserved. Set to 0.

interpassProcessingDoneOut
A pointer to a `Boolean` that will be set to `FALSE` if this function should be called again, `TRUE` if not.

requestedNextPassModeFlagsOut
A pointer to `ICMCompressionPassModeFlags` that will be set to the codec's recommended mode flags for the next pass. `kICMCompressionPassMode_OutputEncodedFrames` will be set only if it recommends that the next pass be the final one:

`kICMCompressionPassMode_OutputEncodedFrames = 1L<<0`
Output encoded frames.

`kICMCompressionPassMode_NoSourceFrames = 1L<<1`
The client need not provide source frame buffers.

`kICMCompressionPassMode_WriteToMultiPassStorage = 1L<<2`
The compressor may write private data to multipass storage.

`kICMCompressionPassMode_ReadFromMultiPassStorage = 1L<<3`
The compressor may read private data from multipass storage.

Return Value
An error code. Returns `noErr` if there is no error.

Discussion
Call this function repeatedly until the compressor sets *interpassProcessingDoneOut* to `TRUE` to indicate that it is done with this round of interpass processing. When done, the compressor will indicate its preferred mode for the next pass. At this point the client may choose to begin an encoding pass, by OR-combining the `kICMCompressionPassMode_OutputEncodedFrames` flag, regardless of the compressor's request.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMCompressionSessionRelease

Decrements the retain count of a compression session.

```
void ICMCompressionSessionRelease (
 ICMCompressionSessionRef    session );
```

Parameters

session
A compression session reference. This reference is returned by `ICMCompressionSessionCreate` (page 106). If you pass `NULL`, nothing happens.

Discussion
If the retain count drops to 0, the session is disposed.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMCompressionSessionRetain

Increments the retain count of a compression session.

```
ICMCompressionSessionRef ICMCompressionSessionRetain (
 ICMCompressionSessionRef    session );
```

Parameters

`session`

A compression session reference. This reference is returned by `ICMCompressionSessionCreate` (page 106). If you pass `NULL`, nothing happens.

Return Value
A reference to the object passed in *`session`*, for convenience.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMCompressionSessionSetProperty

Sets the value of a specific property of a compression session.

```
OSStatus ICMCompressionSessionSetProperty (
 ICMCompressionSessionRef   session,
 ComponentPropertyClass     inPropClass,
 ComponentPropertyID        inPropID,
 ByteCount                  inPropValueSize,
 ConstComponentValuePtr     inPropValueAddress );
```

Parameters

`session`

A compression session reference. This reference is returned by `ICMCompressionSessionCreate` (page 106).

`inPropClass`

Pass the following constant to define the property class:

`kComponentPropertyClassPropertyInfo = 'pnfo'`

The property information class.

inPropID

Pass one of these constants to define the property ID:

`kComponentPropertyInfoList = 'list'`

An array of `CFData` values, one for each property.

`kComponentPropertyCacheSeed = 'seed'`

A property cache seed value.

`kComponentPropertyCacheFlags = 'flgs'`

One of the `kComponentPropertyCache` flags:

`kComponentPropertyCacheFlagNotPersistent`

Property metadata should not be saved in persistent cache.

`kComponentPropertyCacheFlagIsDynamic`

Property metadata should not cached at all.

`kComponentPropertyExtendedInfo = 'meta'`

A `CFDictionary` with extended property information.

inPropValueSize

The size of the property value to be set.

inPropValueAddress

A pointer to the value of the property to be set.

Return Value
An error code. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMCompressionSessionSupportsMultiPassEncoding

Queries whether a compression session supports multipass encoding.

```
Boolean ICMCompressionSessionSupportsMultiPassEncoding (
 ICMCompressionSessionRef       session,
 UInt32                         multiPassStyleFlags,
 ICMCompressionPassModeFlags    *firstPassModeFlagsOut );
```

Parameters

session

A compression session reference. This reference is returned by `ICMCompressionSessionCreate` (page 106).

multiPassStyleFlags

Reserved; set to 0.

firstPassModeFlagsOut

A pointer to a variable to receive the session's requested mode flags for the first pass. The client may modify these flags, but should not set `kICMCompressionPassMode_NoSourceFrames`. Pass `NULL` if you do not want this information.

Return Value

Returns `TRUE` if the compression session supports multipass encoding, `FALSE` otherwise.

Discussion

Even if this function returns `FALSE`, if you passed `TRUE` to `ICMCompressionSessionOptionsSetMultiPass`, you must call `ICMCompressionSessionBeginPass` and `ICMCompressionSessionEndPass`.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMCompressorSessionDropFrame

Called by a compressor to notify the ICM that a source frame has been dropped and will not contribute to any encoded frames.

```
OSStatus ICMCompressorSessionDropFrame (
 ICMCompressorSessionRef       session,
 ICMCompressorSourceFrameRef   sourceFrame );
```

Parameters

session

A reference to the compression session between the ICM and an image compressor component.

sourceFrame

A reference to a frame that has been passed in *sourceFrameRefCon* to `ICMCompressionSessionEncodeFrame` (page 108). If you pass `NULL`, nothing happens.

Return Value

An error code. Returns `noErr` if there is no error.

Discussion

Calling this function does not automatically release the source frame; if the compressor called `ICMCompressorSourceFrameRetain` it should still call `ICMCompressorSourceFrameRelease`.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMCompressorSessionEmitEncodedFrame

Called by a compressor to output an encoded frame corresponding to one or more source frames.

```
OSStatus ICMCompressorSessionEmitEncodedFrame (
 ICMCompressorSessionRef       session,
 ICMMutableEncodedFrameRef     encodedFrame,
 long                          numberOfSourceFrames,
 ICMCompressorSourceFrameRef   sourceFrames[] );
```

Parameters

*sessio*n

A reference to the compression session between the ICM and an image compressor component.

encodedFrame

A reference to an encoded frame object with write capabilities.

numberOfSourceFrames

The number of source frames encoded in the encoded frame.

sourceFrames

References to frames that have been passed in *sourceFrameRefCon* to `ICMCompressionSessionEncodeFrame` (page 108).

Return Value

An error code. Returns `noErr` if there is no error.

Discussion

Encoded frames may correspond to more than one source frame only if `allowFrameTimeChanges` is set in the compression session's `compressionSessionOptions`.

After calling this function, the compressor should release the encoded frame by calling `ICMEncodedFrameRelease`. Calling this function does not automatically release the source frames; if the compressor called `ICMCompressorSourceFrameRetain` it should still call `ICMCompressorSourceFrameRelease`.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMCompressorSourceFrameGetDisplayNumber

Retrieves a source frames display number.

```
long ICMCompressorSourceFrameGetDisplayNumber (
 ICMCompressorSourceFrameRef   sourceFrame );
```

Parameters

sourceFrame

A reference to a frame that has been passed in *sourceFrameRefCon* to `ICMCompressionSessionEncodeFrame` (page 108).

Return Value
The display number of the source frame.

Discussion
The ICM tags source frames with display numbers in the order that they are passed to `ICMCompressionSessionEncodeFrame`. The first display number is 1. Compressors may compare these numbers to work out whether prediction is forward or backward, even when display times are not provided.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMCompressorSourceFrameGetDisplayTimeStampAndDuration

Retrieves the display time stamp and duration of a source frame.

```
OSStatus ICMCompressorSourceFrameGetDisplayTimeStampAndDuration (
 ICMCompressorSourceFrameRef    sourceFrame,
 TimeValue64                    *displayTimeStampOut,
 TimeValue64                    *displayDurationOut,
 TimeScale                      *timeScaleOut,
 ICMValidTimeFlags              *validTimeFlagsOut );
```

Parameters

sourceFrame
A reference to a frame that has been passed in *sourceFrameRefCon* to `ICMCompressionSessionEncodeFrame` (page 108).

displayTimeStampOut
A pointer to the source frame's display time stamp.

displayDurationOut
A pointer to the source frame's display duration.

timeScaleOut
A pointer to the source frame's display time scale.

validTimeFlagsOut
A pointer to one of these display time flags for the source frame:

`kICMValidTime_DisplayTimeStampIsValid = 1L<<0`
The value of *displayTimeStamp* is valid.

`kICMValidTime_DisplayDurationIsValid = 1L<<1`
The value of *displayDuration* is valid.

Return Value
An error code. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMCompressorSourceFrameGetFrameOptions

Retrieves the frame compression options for a source frame.

```
ICMCompressionFrameOptionsRef ICMCompressorSourceFrameGetFrameOptions (
 ICMCompressorSourceFrameRef   sourceFrame );
```

Parameters

sourceFrame
A reference to a frame that has been passed in *sourceFrameRefCon* to `ICMCompressionSessionEncodeFrame` (page 108).

Return Value
A compression session frame options reference representing options for this frame. A frame options object is created by `ICMCompressionFrameOptionsCreate` (page 96).

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMCompressorSourceFrameGetPixelBuffer

Retrieves a source frames pixel buffer.

```
CVPixelBufferRef ICMCompressorSourceFrameGetPixelBuffer (
 ICMCompressorSourceFrameRef   sourceFrame );
```

Parameters

sourceFrame
A reference to a frame that has been passed in *sourceFrameRefCon* to `ICMCompressionSessionEncodeFrame` (page 108).

Return Value
A reference to the pixel buffer containing the source frame's image being compressed.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMCompressorSourceFrameGetTypeID

Returns the type ID for the current source frame object.

```
CFTypeID ICMCompressorSourceFrameGetTypeID ( void );
```

Return Value
A `CFTypeID` value.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMCompressorSourceFrameRelease

Decrements the retain count of a source frame object.

```
void ICMCompressorSourceFrameRelease (
 ICMCompressorSourceFrameRef    sourceFrame );
```

Parameters

`sourceFrame`
: A reference to a frame that has been passed in *`sourceFrameRefCon`* to `ICMCompressionSessionEncodeFrame` (page 108). If you pass `NULL`, nothing happens.

Discussion
If the retain count drops to 0, the object is disposed.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMCompressorSourceFrameRetain

Increments the retain count of a source frame object.

```
ICMCompressorSourceFrameRef ICMCompressorSourceFrameRetain (
 ICMCompressorSourceFrameRef    sourceFrame );
```

Parameters

`sourceFrame`
: A reference to a frame that has been passed in *`sourceFrameRefCon`* to `ICMCompressionSessionEncodeFrame` (page 108). If you pass `NULL`, nothing happens.

Return Value
A reference to the object passed in *sourceFrame*, for convenience.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMDecompressionFrameOptionsCreate

Creates a frame decompression options object.

```
OSStatus ICMDecompressionFrameOptionsCreate (
 CFAllocatorRef                     allocator,
 ICMDecompressionFrameOptionsRef    *options );
```

Parameters

allocator

An allocator. Pass `NULL` to use the default allocator.

options

On return, a reference to a frame decompression options object.

Return Value
An error code. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMDecompressionFrameOptionsCreateCopy

Copies a frame decompression options object.

```
OSStatus ICMDecompressionFrameOptionsCreateCopy (
 CFAllocatorRef                     allocator,
 ICMDecompressionFrameOptionsRef    originalOptions,
 ICMDecompressionFrameOptionsRef    *copiedOptions );
```

Parameters

allocator

An allocator. Pass `NULL` to use the default allocator.

originalOptions

A reference to a frame decompression options object. You can create this object by calling `ICMDecompressionFrameOptionsCreate` (page 134).

copiedOptions

On return, a reference to a copy of the frame decompression options object passed in *originalOptions*.

Return Value

An error code. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMDecompressionFrameOptionsGetProperty

Retrieves the value of a specific property of a decompression frame options object.

```
OSStatus ICMDecompressionFrameOptionsGetProperty (
 ICMDecompressionFrameOptionsRef   options,
 ComponentPropertyClass            inPropClass,
 ComponentPropertyID               inPropID,
 ByteCount                         inPropValueSize,
 ComponentValuePtr                 outPropValueAddress,
 ByteCount                         *outPropValueSizeUsed );
```

Parameters

options

A decompression frame options reference. This reference is returned by `ICMDecompressionFrameOptionsCreate` (page 134).

inPropClass

Pass the following constant to define the property class:

`kComponentPropertyClassPropertyInfo = 'pnfo'`

The property information class.

inPropID

Pass one of these constants to define the property ID:

`kComponentPropertyInfoList = 'list'`

An array of `CFData` values, one for each property.

`kComponentPropertyCacheSeed = 'seed'`

A property cache seed value.

`kComponentPropertyCacheFlags = 'flgs'`

One of the `kComponentPropertyCache` flags:

`kComponentPropertyCacheFlagNotPersistent`

Property metadata should not be saved in persistent cache.

`kComponentPropertyCacheFlagIsDynamic`

Property metadata should not cached at all.

`kComponentPropertyExtendedInfo = 'meta'`

A `CFDictionary` with extended property information.

outPropType

A pointer to the type of the returned property's value.

outPropValueAddress

A pointer to a variable to receive the returned property's value.

outPropValueSizeUsed

On return, a pointer to the number of bytes actually used to store the property.

Return Value

An error code. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMDecompressionFrameOptionsGetPropertyInfo

Retrieves information about properties of a decompression frame options object.

```
OSStatus ICMDecompressionFrameOptionsGetPropertyInfo (
 ICMDecompressionFrameOptionsRef   options,
 ComponentPropertyClass            inPropClass,
 ComponentPropertyID               inPropID,
 ComponentValueType                *outPropType,
 ByteCount                         *outPropValueSize,
 UInt32                            *outPropertyFlags );
```

Parameters

options

A decompression frame options reference. This reference is returned by `ICMDecompressionFrameOptionsCreate` (page 134).

inPropClass

Pass the following constant to define the property class:

`kComponentPropertyClassPropertyInfo = 'pnfo'`

The property information class.

inPropID

Pass one of these constants to define the property ID:

`kComponentPropertyInfoList = 'list'`

An array of `CFData` values, one for each property.

`kComponentPropertyCacheSeed = 'seed'`

A property cache seed value.

`kComponentPropertyCacheFlags = 'flgs'`

One of the `kComponentPropertyCache` flags:

`kComponentPropertyCacheFlagNotPersistent`

Property metadata should not be saved in persistent cache.

`kComponentPropertyCacheFlagIsDynamic`

Property metadata should not cached at all.

`kComponentPropertyExtendedInfo = 'meta'`

A `CFDictionary` with extended property information.

outPropType

A pointer to the type of the returned property's value.

outPropValueSize

A pointer to the size of the returned property's value.

outPropFlags

On return, a pointer to flags representing the requested information about the frame option's property.

Return Value

An error code. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMDecompressionFrameOptionsGetTypeID

Returns the type ID for the current frame decompression options object.

```
CFTypeID ICMDecompressionFrameOptionsGetTypeID ( void );
```

Return Value
A `CFTypeID` value.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMDecompressionFrameOptionsRelease

Decrements the retain count of a frame decompression options object.

```
void ICMDecompressionFrameOptionsRelease (
 ICMDecompressionFrameOptionsRef    options );
```

Parameters

`options`
A reference to a frame decompression options object. You can create this object by calling `ICMDecompressionFrameOptionsCreate` (page 134). If you pass `NULL`, nothing happens.

Discussion
If the retain count drops to 0, the object is disposed.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMDecompressionFrameOptionsRetain

Increments the retain count of a frame decompression options object.

```
ICMDecompressionFrameOptionsRef ICMDecompressionFrameOptionsRetain (
 ICMDecompressionFrameOptionsRef    options );
```

Parameters

options

A reference to a frame decompression options object. You can create this object by calling `ICMDecompressionFrameOptionsCreate` (page 134). If you pass `NULL`, nothing happens.

Return Value

A reference to the frame decompression options object passed in *options*, for convenience.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMDecompressionFrameOptionsSetProperty

Sets the value of a specific property of a decompression frame options object.

```
OSStatus ICMDecompressionFrameOptionsSetProperty (
 ICMDecompressionFrameOptionsRef   options,
 ComponentPropertyClass            inPropClass,
 ComponentPropertyID               inPropID,
 ByteCount                         inPropValueSize,
 ConstComponentValuePtr            inPropValueAddress );
```

Parameters

options

A decompression frame options reference. This reference is returned by `ICMDecompressionFrameOptionsCreate` (page 134).

inPropClass

Pass the following constant to define the property class:

`kComponentPropertyClassPropertyInfo = 'pnfo'`

The property information class.

inPropID

Pass one of these constants to define the property ID:

`kComponentPropertyInfoList = 'list'`

An array of `CFData` values, one for each property.

`kComponentPropertyCacheSeed = 'seed'`

A property cache seed value.

`kComponentPropertyCacheFlags = 'flgs'`

One of the `kComponentPropertyCache` flags:

`kComponentPropertyCacheFlagNotPersistent`

Property metadata should not be saved in persistent cache.

`kComponentPropertyCacheFlagIsDynamic`

Property metadata should not cached at all.

`kComponentPropertyExtendedInfo = 'meta'`

A `CFDictionary` with extended property information.

inPropValueSize

The size of the property value to be set.

inPropValueAddress

A pointer to the value of the property to be set.

Return Value

An error code. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMDecompressionSessionCreate

Creates a session for decompressing video frames.

```
OSStatus ICMDecompressionSessionCreate (
 CFAllocatorRef                      allocator,
 ImageDescriptionHandle              desc,
 ICMDecompressionSessionOptionsRef   decompressionOptions,
 CFDictionaryRef                     destinationPixelBufferAttributes,
 ICMDecompressionTrackingCallbackRecord    *trackingCallback,
 ICMDecompressionSessionRef                *decompressionSessionOut );
```

Parameters

allocator

An allocator for the session. Pass `NULL` to use the default allocator.

desc
: An image description for the source frames.

decompressionOptions
: A decompression session options reference. This reference is returned by `ICMDecompressionSessionOptionsCreate` (page 146). The session will retain the object. You may change some options during the session by modifying the object. You may also pass `NULL`.

destinationPixelBufferAttributes
: Requirements for emitted pixel buffers. You may pass `NULL`.

trackingCallback
: A pointer to a structure that designates a callback to be called for information about queued frames and pixel buffers containing decompressed frames. See `ICMDecompressionTrackingCallbackRecord` (page 258) and `ICMDecompressionTrackingCallbackProc` (page 254).

decompressionSessionOut
: A pointer to a variable to receive a reference to the new decompression session.

Return Value
An error code. Returns `noErr` if there is no error.

Discussion
Frames are returned through calls to the callback pointed to by `trackingCallback`.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMDecompressionSessionCreateForVisualContext

Creates a session for decompressing video frames.

```
OSStatus ICMDecompressionSessionCreateForVisualContext (
 CFAllocatorRef allocator,
 /*can be NULL */ ImageDescriptionHandle desc,
 ICMDecompressionSessionOptionsRef decompressionOptions,
 /*can be NULL */ QTVisualContextRef visualContext,
 ICMDecompressionTrackingCallbackRecord *trackingCallback,
 ICMDecompressionSessionRef *decompressionSessionOut );
```

Parameters

allocator
: An allocator for the session. Pass `NULL` to use the default allocator.

desc
: An image description for the source frames.

decompressionOptions
: Options for the session. The session will retain this options object. You may change some options during the session by modifying the object.

visualContext

The target visual context.

trackingCallback

The callback to be called with information about queued frames, and pixel buffers containing the decompressed frames.

decompressionSessionOut

Points to a variable to receive the new decompression session.

Return Value

An error code. Returns `noErr` if there is no error.

Discussion

Frames will be output to a visual context. If desired, the *trackingCallback* may attach additional data to pixel buffers before they are sent to the visual context.

Version Notes

Introduced in QuickTime 7

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMDecompressionSessionDecodeFrame

Queues a frame for decompression.

```
OSStatus ICMDecompressionSessionDecodeFrame (
 ICMDecompressionSessionRef         session,
 const UInt8                        *data,
 ByteCount                          dataSize,
 ICMDecompressionFrameOptionsRef    frameOptions,
 const ICMFrameTimeRecord           *frameTime,
 void                               *sourceFrameRefCon );
```

Parameters

session

A decompression session reference. This reference is returned by `ICMDecompressionSessionCreate` (page 140).

data

A pointer to the compressed data for this frame. The data must remain in this location until `ICMDecompressionTrackingCallbackProc` (page 254) is called with the `kICMDecompressionTracking_ReleaseSourceData` flag set in *decompressionTrackingFlags*.

dataSize

The number of bytes of compressed data. You may not pass 0 in this parameter.

frameOptions

A reference to a frame decompression options object containing options for this frame. You can create this object by calling `ICMDecompressionFrameOptionsCreate` (page 134).

frameTime

A pointer to a structure describing the frame's timing information.

sourceFrameRefCon

Your reference value for the frame.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMDecompressionSessionFlush

Flushes the frames queued for a decompression session.

```
OSStatus ICMDecompressionSessionFlush (
 ICMDecompressionSessionRef   session );
```

Parameters

session

A decompression session reference. This reference is returned by `ICMDecompressionSessionCreate` (page 140).

Return Value

An error code. Returns `noErr` if there is no error.

Discussion

The tracking callback will be called for each frame with the result –1.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMDecompressionSessionGetProperty

Retrieves the value of a specific property of a decompression session.

```
OSStatus ICMDecompressionSessionGetProperty (
 ICMDecompressionSessionRef   session,
 ComponentPropertyClass       inPropClass,
 ComponentPropertyID          inPropID,
 ByteCount                    inPropValueSize,
 ComponentValuePtr            outPropValueAddress,
 ByteCount                    *outPropValueSizeUsed );
```

Parameters

session

A decompression session reference. This reference is returned by `ICMDecompressionSessionCreate` (page 140).

inPropClass

Pass the following constant to define the property class:

`kComponentPropertyClassPropertyInfo = 'pnfo'`

The property information class.

inPropID

Pass one of these constants to define the property ID:

`kComponentPropertyInfoList = 'list'`

An array of `CFData` values, one for each property.

`kComponentPropertyCacheSeed = 'seed'`

A property cache seed value.

`kComponentPropertyCacheFlags = 'flgs'`

One of the `kComponentPropertyCache` flags:

`kComponentPropertyCacheFlagNotPersistent`

Property metadata should not be saved in persistent cache.

`kComponentPropertyCacheFlagIsDynamic`

Property metadata should not cached at all.

`kComponentPropertyExtendedInfo = 'meta'`

A `CFDictionary` with extended property information.

outPropType

A pointer to the type of the returned property's value.

outPropValueAddress

A pointer to a variable to receive the returned property's value.

outPropValueSizeUsed

On return, a pointer to the number of bytes actually used to store the property.

Return Value

An error code. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMDecompressionSessionGetPropertyInfo

Retrieves information about the properties of a decompression session.

```
OSStatus ICMDecompressionSessionGetPropertyInfo (
 ICMDecompressionSessionRef   session,
 ComponentPropertyClass       inPropClass,
 ComponentPropertyID          inPropID,
 ComponentValueType           *outPropType,
 ByteCount                    *outPropValueSize,
 UInt32                       *outPropertyFlags );
```

Parameters

session

A decompression session reference. This reference is returned by `ICMDecompressionSessionCreate` (page 140).

inPropClass

Pass the following constant to define the property class:

`kComponentPropertyClassPropertyInfo = 'pnfo'`

The property information class.

inPropID

Pass one of these constants to define the property ID:

`kComponentPropertyInfoList = 'list'`

An array of `CFData` values, one for each property.

`kComponentPropertyCacheSeed = 'seed'`

A property cache seed value.

`kComponentPropertyCacheFlags = 'flgs'`

One of the `kComponentPropertyCache` flags:

`kComponentPropertyCacheFlagNotPersistent`

Property metadata should not be saved in persistent cache.

`kComponentPropertyCacheFlagIsDynamic`

Property metadata should not cached at all.

`kComponentPropertyExtendedInfo = 'meta'`

A `CFDictionary` with extended property information.

outPropType

A pointer to the type of the returned property's value.

outPropValueSize

A pointer to the size of the returned property's value.

outPropFlags

On return, a pointer to flags representing the requested information about the property.

Return Value

An error code. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMDecompressionSessionGetTypeID

Returns the type ID for the current decompression session.

```
CFTypeID ICMDecompressionSessionGetTypeID ( void );
```

Return Value
A `CFTypeID` value.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMDecompressionSessionOptionsCreate

Creates a decompression session options object.

```
OSStatus ICMDecompressionSessionOptionsCreate (
 CFAllocatorRef                      allocator,
 ICMDecompressionSessionOptionsRef   *options );
```

Parameters

`allocator`
An allocator. Pass `NULL` to use the default allocator.

`options`
On return, a reference to a decompression session options object.

Return Value
An error code. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMDecompressionSessionOptionsCreateCopy

Copies a decompression session options object.

```
OSStatus ICMDecompressionSessionOptionsCreateCopy (
 CFAllocatorRef                     allocator,
 ICMDecompressionSessionOptionsRef  originalOptions,
 ICMDecompressionSessionOptionsRef  *copiedOptions );
```

Parameters

allocator

An allocator. Pass `NULL` to use the default allocator.

originalOptions

A decompression session options reference. This reference is returned by `ICMDecompressionSessionOptionsCreate` (page 146).

copiedOptions

On return, a reference to a copy of the decompression session options object passed in *originalOptions*.

Return Value

An error code. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMDecompressionSessionOptionsGetPropertyInfo

Retrieves information about properties of a decompression session options object.

```
OSStatus ICMDecompressionSessionOptionsGetPropertyInfo (
 ICMDecompressionSessionOptionsRef  options,
 ComponentPropertyClass             inPropClass,
 ComponentPropertyID                inPropID,
 ComponentValueType                 *outPropType,
 ByteCount                          *outPropValueSize,
 UInt32                             *outPropertyFlags );
```

Parameters

options

A decompression session options reference. This reference is returned by `ICMDecompressionSessionOptionsCreate` (page 146).

inPropClass

Pass the following constant to define the property class:

`kComponentPropertyClassPropertyInfo = 'pnfo'`

The property information class.

inPropID

Pass one of these constants to define the property ID:

`kComponentPropertyInfoList = 'list'`

An array of `CFData` values, one for each property.

`kComponentPropertyCacheSeed = 'seed'`

A property cache seed value.

`kComponentPropertyCacheFlags = 'flgs'`

One of the `kComponentPropertyCache` flags:

`kComponentPropertyCacheFlagNotPersistent`

Property metadata should not be saved in persistent cache.

`kComponentPropertyCacheFlagIsDynamic`

Property metadata should not cached at all.

`kComponentPropertyExtendedInfo = 'meta'`

A `CFDictionary` with extended property information.

outPropType

A pointer to the type of the returned property's value.

outPropValueSize

A pointer to the size of the returned property's value.

outPropFlags

On return, a pointer to flags representing the requested information about the property.

Return Value
An error code. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMDecompressionSessionOptionsGetProperty

Retrieves the value of a specific property of a decompression session options object.

```
OSStatus ICMDecompressionSessionOptionsGetProperty (
 ICMDecompressionSessionOptionsRef   options,
 ComponentPropertyClass              inPropClass,
 ComponentPropertyID                 inPropID,
 ByteCount                           inPropValueSize,
 ComponentValuePtr                   outPropValueAddress,
 ByteCount                           *outPropValueSizeUsed );
```

Parameters

options

A decompression session options reference. This reference is returned by `ICMDecompressionSessionOptionsCreate` (page 146).

inPropClass

Pass the following constant to define the property class:

`kComponentPropertyClassPropertyInfo = 'pnfo'`

The property information class.

inPropID

Pass one of these constants to define the property ID:

`kComponentPropertyInfoList = 'list'`

An array of `CFData` values, one for each property.

`kComponentPropertyCacheSeed = 'seed'`

A property cache seed value.

`kComponentPropertyCacheFlags = 'flgs'`

One of the `kComponentPropertyCache` flags:

`kComponentPropertyCacheFlagNotPersistent`

Property metadata should not be saved in persistent cache.

`kComponentPropertyCacheFlagIsDynamic`

Property metadata should not cached at all.

`kComponentPropertyExtendedInfo = 'meta'`

A `CFDictionary` with extended property information.

inPropValueSize

The size of the property value to be retrieved.

outPropValueAddress

A pointer to a variable to hold the value of the property.

outPropValueSizeUsed

On return, a pointer to the number of bytes actually used to store the property value.

Return Value

An error code. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMDecompressionSessionOptionsGetTypeID

Returns the type ID for the current decompression session options object.

```
CFTypeID ICMDecompressionSessionOptionsGetTypeID ( void );
```

Return Value

A `CFTypeID` value.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMDecompressionSessionOptionsRelease

Decrements the retain count of a decompression session options object.

```
void ICMDecompressionSessionOptionsRelease (
 ICMDecompressionSessionOptionsRef   options );
```

Parameters

`options`

A reference to a decompression session options object. This reference is returned by `ICMDecompressionSessionOptionsCreate` (page 146). If you pass `NULL`, nothing happens.

Discussion

If the retain count drops to 0, the object is disposed.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMDecompressionSessionOptionsRetain

Increments the retain count of a decompression session options object.

```
ICMDecompressionSessionOptionsRef ICMDecompressionSessionOptionsRetain (
 ICMDecompressionSessionOptionsRef   options );
```

Parameters

options

A reference to a decompression session options object. This reference is returned by `ICMDecompressionSessionOptionsCreate` (page 146). If you pass `NULL`, nothing happens.

Return Value

A copy of the object reference passed in *options*, for convenience.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMDecompressionSessionOptionsSetProperty

Sets the value of a specific property of a decompression session options object.

```
OSStatus ICMDecompressionSessionOptionsSetProperty (
 ICMDecompressionSessionOptionsRef   options,
 ComponentPropertyClass              inPropClass,
 ComponentPropertyID                 inPropID,
 ByteCount                           inPropValueSize,
 ConstComponentValuePtr              inPropValueAddress );
```

Parameters

options

A decompression session options reference. This reference is returned by `ICMDecompressionSessionOptionsCreate` (page 146).

inPropClass

Pass the following constant to define the property class:

`kComponentPropertyClassPropertyInfo = 'pnfo'`

The property information class.

inPropID

Pass one of these constants to define the property ID:

`kComponentPropertyInfoList = 'list'`

An array of `CFData` values, one for each property.

`kComponentPropertyCacheSeed = 'seed'`

A property cache seed value.

`kComponentPropertyCacheFlags = 'flgs'`

One of the `kComponentPropertyCache` flags:

`kComponentPropertyCacheFlagNotPersistent`

Property metadata should not be saved in persistent cache.

`kComponentPropertyCacheFlagIsDynamic`

Property metadata should not cached at all.

`kComponentPropertyExtendedInfo = 'meta'`

A `CFDictionary` with extended property information.

inPropValueSize

The size of the property value to be set.

inPropValueAddress

A pointer to the value of the property to be set.

Return Value

An error code. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMDecompressionSessionRetain

Increments the retain count of a decompression session.

```
ICMDecompressionSessionRef ICMDecompressionSessionRetain (
 ICMDecompressionSessionRef    session );
```

Parameters

session

A decompression session reference. This reference is returned by `ICMDecompressionSessionCreate` (page 140). If you pass `NULL`, nothing happens.

Return Value

A copy of the reference passed in *session*, for convenience.

Version Notes

Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMDecompressionSessionRelease

Decrements the retain count of a decompression session.

```
void ICMDecompressionSessionRelease (
 ICMDecompressionSessionRef    session );
```

Parameters

`session`
: A decompression session reference. This reference is returned by `ICMDecompressionSessionCreate` (page 140). If you pass `NULL`, nothing happens.

Discussion
If the retain count drops to 0, the object is disposed.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMDecompressionSessionSetNonScheduledDisplayDirection

Sets the direction for non-scheduled display time.

```
OSStatus ICMDecompressionSessionSetNonScheduledDisplayDirection (
 ICMDecompressionSessionRef   session,
 Fixed                        rate );
```

Parameters

`session`
: A decompression session reference. This reference is returned by `ICMDecompressionSessionCreate` (page 140).

`rate`
: The display direction. Negative values represent backward display and positive values represent forward display.

Return Value
An error code. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMDecompressionSessionSetNonScheduledDisplayTime

Sets the display time for a decompression session, and requests display of the non-scheduled queued frame at that display time, if there is one.

```
OSStatus ICMDecompressionSessionSetNonScheduledDisplayTime (
 ICMDecompressionSessionRef   session,
 TimeValue64                  displayTime,
 TimeScale                    displayTimeScale,
 UInt32                       flags );
```

Parameters

session
A decompression session reference. This reference is returned by `ICMDecompressionSessionCreate` (page 140).

displayTime
A display time. Usually this is the display time of a non-scheduled queued frame.

displayTimeScale
The timescale according to which *displayTime* should be interpreted.

flags
Reserved; set to 0.

Return Value
An error code. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMDecompressionSessionSetProperty

Sets the value of a specific property of a decompression session.

```
OSStatus ICMDecompressionSessionSetProperty (
 ICMDecompressionSessionRef   session,
 ComponentPropertyClass       inPropClass,
 ComponentPropertyID          inPropID,
 ByteCount                    inPropValueSize,
 ConstComponentValuePtr       inPropValueAddress );
```

Parameters

session
A decompression session reference. This reference is returned by `ICMDecompressionSessionCreate` (page 140).

inPropClass

Pass the following constant to define the property class:

`kComponentPropertyClassPropertyInfo = 'pnfo'`

The property information class.

inPropID

Pass one of these constants to define the property ID:

`kComponentPropertyInfoList = 'list'`

An array of `CFData` values, one for each property.

`kComponentPropertyCacheSeed = 'seed'`

A property cache seed value.

`kComponentPropertyCacheFlags = 'flgs'`

One of the `kComponentPropertyCache` flags:

`kComponentPropertyCacheFlagNotPersistent`

Property metadata should not be saved in persistent cache.

`kComponentPropertyCacheFlagIsDynamic`

Property metadata should not cached at all.

`kComponentPropertyExtendedInfo = 'meta'`

A `CFDictionary` with extended property information.

inPropValueSize

The size in bytes of the property's value.

inPropValueAddress

A pointer to the property value to be set.

Return Value

An error code. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMEncodedFrameGetBufferSize

Gets the size of an encoded frame's data buffer.

```
ByteCount ICMEncodedFrameGetBufferSize (
 ICMEncodedFrameRef   frame );
```

Parameters

frame

A reference to an encoded frame object.

Return Value
The physical size in bytes of the encoded frame's data buffer.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMEncodedFrameCreateMutable

Called by a compressor to create an encoded-frame token corresponding to a given source frame.

```
OSStatus ICMEncodedFrameCreateMutable (
 ICMCompressorSessionRef       session,
 ICMCompressorSourceFrameRef   sourceFrame,
 ByteCount                     bufferSize,
 ICMMutableEncodedFrameRef     *frameOut );
```

Parameters

session
A reference to the compression session between the ICM and an image compressor component.

sourceFrame
A reference to a frame that has been passed in *sourceFrameRefCon* to `ICMCompressionSessionEncodeFrame` (page 108).

bufferSize
The size of the frame buffer in bytes.

frameOut
On return, a reference to an encoded frame object with write capabilities.

Return Value
An error code. Returns `noErr` if there is no error.

Discussion
The encoded frame will initially show 0 for `mediaSampleFlags`; if the frame is not a key frame, the compressor must call `ICMEncodedFrameSetMediaSampleFlags` to set `mediaSampleNotSync`. If the frame is droppable, the compressor should set `mediaSampleDroppable`. If the frame is a partial key frame, the compressor should set `mediaSamplePartialSync`.

The encoded frame will initially have undefined `decodeTimeStamp` and `decodeDuration` values. The compressor may set these directly by calling `ICMEncodedFrameSetDecodeTimeStamp` and `ICMEncodedFrameSetDecodeDuration`. If these are not set by the compressor, the ICM will try to derive values for them.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMEncodedFrameGetDataPtr

Gets the data buffer for an encoded frame.

```
UInt8 *ICMEncodedFrameGetDataPtr (
 ICMEncodedFrameRef   frame );
```

Parameters

`frame`
A reference to an encoded frame object.

Return Value
A pointer to the object's data buffer.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMEncodedFrameGetDataSize

Gets the data size of the compressed frame in an encoded frame's buffer.

```
ByteCount ICMEncodedFrameGetDataSize (
 ICMEncodedFrameRef   frame );
```

Parameters

`frame`
A reference to an encoded frame object.

Return Value
The logical size in bytes of the encoded frame's data buffer, which may be less than the physical size of the buffer.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMEncodedFrameGetDecodeDuration

Retrieves an encoded frame's decode duration.

```
TimeValue64 ICMEncodedFrameGetDecodeDuration (
 ICMEncodedFrameRef   frame );
```

Parameters

frame

A reference to an encoded frame object.

Return Value

The encoded frame's decode duration.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMEncodedFrameGetDecodeNumber

Retrieves the decode number of an encoded frame.

```
UInt32 ICMEncodedFrameGetDecodeNumber (
 ICMEncodedFrameRef   frame );
```

Parameters

frame

A reference to an encoded frame object.

Return Value

The decode number of the encoded frame.

Discussion

The ICM automatically stamps ascending decode numbers on frames after the compressor emits them. The first decode number in session is 1. Compressors should not call this function.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMEncodedFrameGetDecodeTimeStamp

Retrieves an encoded frame's decode time stamp.

```
TimeValue64 ICMEncodedFrameGetDecodeTimeStamp (
 ICMEncodedFrameRef   frame );
```

Parameters

frame

A reference to an encoded frame object.

Return Value

The encoded frame's decode time stamp.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMEncodedFrameGetDisplayDuration

Retrieves an encoded frame's display duration.

```
TimeValue64 ICMEncodedFrameGetDisplayDuration (
 ICMEncodedFrameRef   frame );
```

Parameters

frame

A reference to an encoded frame object.

Return Value

The encoded frame's display duration.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMEncodedFrameGetDisplayOffset

Retrieves an encoded frame's display offset.

```
TimeValue64 ICMEncodedFrameGetDisplayOffset (
 ICMEncodedFrameRef   frame );
```

Parameters

frame

A reference to an encoded frame object.

Return Value

The encoded frame's display offset. This is the time offset from decode time stamp to display time stamp.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMEncodedFrameGetDisplayTimeStamp

Retrieves an encoded frame's display time stamp.

```
TimeValue64 ICMEncodedFrameGetDisplayTimeStamp (
 ICMEncodedFrameRef   frame );
```

Parameters

`frame`
A reference to an encoded frame object.

Return Value
The encoded frame's display time stamp.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMEncodedFrameGetFrameType

Retrieves the `frame` type for an encoded frame.

```
ICMFrameType ICMEncodedFrameGetFrameType (
 ICMEncodedFrameRef   frame );
```

Parameters

`frame`
A reference to an encoded frame object.

Return Value
The encoded frame's frame type (see below).

Discussion
This function returns one of these values:

`kICMFrameType_I = 'I'`
An I frame.

`kICMFrameType_P = 'P'`
A P frame.

`kICMFrameType_B = 'B'`
: A B frame.

`kICMFrameType_Unknown = 0`
: A frame of unknown type.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMEncodedFrameGetImageDescription

Retrieves the image description of an encoded frame.

```
OSStatus ICMEncodedFrameGetImageDescription (
 ICMEncodedFrameRef       frame,
 ImageDescriptionHandle   *imageDescOut );
```

Parameters

`frame`
: A reference to an encoded frame object.

`imageDescOut`
: A pointer to a handle containing the encoded frame's image description. The caller should not dispose of this handle.

Return Value
An error code. Returns `noErr` if there is no error.

Discussion
This function returns the same image description handle as `ICMCompressionSessionGetImageDescription`.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMEncodedFrameGetMediaSampleFlags

Retrieves the media sample flags for an encoded frame.

```
MediaSampleFlags ICMEncodedFrameGetMediaSampleFlags (
 ICMEncodedFrameRef   frame );
```

Parameters

frame

A reference to an encoded frame object.

Return Value

The object's media sample flags. These flags are listed in the header file `Movies.h`.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMEncodedFrameGetSimilarity

Retrieves the similarity value for an encoded frame.

```
Float32 ICMEncodedFrameGetSimilarity (
 ICMEncodedFrameRef   frame );
```

Parameters

frame

A reference to an encoded frame object.

Return Value

The encoded frame's similarity value. 1.0 means identical; 0.0 means not at all alike. The default value is –1.0, which means unknown.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMEncodedFrameGetSourceFrameRefCon

Retrieves the reference value of an encoded frame's source frame.

```
void *ICMEncodedFrameGetSourceFrameRefCon (
 ICMEncodedFrameRef   frame );
```

Parameters

frame

A reference to an encoded frame object.

Discussion
The source frame's reference value is copied from the session's *`sourceFrameRefCon`* parameter that was passed to `ICMCompressionSessionEncodeFrame` (page 108).

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMEncodedFrameGetTimeScale

Retrieves the timescale of an encoded frame.

```
TimeScale ICMEncodedFrameGetTimeScale (
 ICMEncodedFrameRef   frame );
```

Parameters

`frame`
    A reference to an encoded frame object.

Return Value
The time scale of an encoded frame. This is always the same as the time scale of the compression session.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMEncodedFrameGetTypeID

Returns the type ID for the current encoded frame object.

```
CFTypeID ICMEncodedFrameGetTypeID ( void );
```

Return Value
A `CFTypeID` value.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMEncodedFrameGetValidTimeFlags

Retrieves an encoded frame's flags indicating which of its time stamps and durations are valid.

```
ICMValidTimeFlags ICMEncodedFrameGetValidTimeFlags (
 ICMEncodedFrameRef   frame );
```

Parameters

frame

A reference to an encoded frame object.

Return Value

One of the constants listed below.

Discussion

This function returns one of these values:

`kICMValidTime_DisplayTimeStampIsValid = 1L<<0`

The value of *displayTimeStamp* is valid.

`kICMValidTime_DisplayDurationIsValid = 1L<<1`

The value of *displayDuration* is valid.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMEncodedFrameRelease

Decrements the retain count of an encoded frame object.

```
void ICMEncodedFrameRelease (
 ICMEncodedFrameRef    frame );
```

Parameters

frame

A reference to an encoded frame object. If you pass `NULL`, nothing happens.

Discussion

If the retain count drops to 0, the object is disposed.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMEncodedFrameRetain

Increments the retain count of an encoded frame object.

```
ICMEncodedFrameRef ICMEncodedFrameRetain (
 ICMEncodedFrameRef    frame );
```

Parameters

frame

A reference to an encoded frame object. If you pass NULL, nothing happens.

Return Value

A reference to the object passed in *frame*, for convenience.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: ImageCompression.h

Declared In

ImageCompression.h

ICMEncodedFrameSetDataSize

Sets the data size of the compressed frame in an encoded frame's buffer.

```
OSStatus ICMEncodedFrameSetDataSize (
 ICMMutableEncodedFrameRef   frame,
 ByteCount                   dataSize );
```

Parameters

frame

A reference to an encoded frame object with write capabilities.

dataSize

The data size of the compressed frame in the encoded frame object's buffer.

Return Value

An error code. Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: ImageCompression.h

Declared In

ImageCompression.h

ICMEncodedFrameSetDecodeDuration

Sets an encoded frame's decode duration.

```
OSStatus ICMEncodedFrameSetDecodeDuration (
 ICMMutableEncodedFrameRef   frame,
 TimeValue64                 decodeDuration );
```

Parameters

frame

A reference to an encoded frame object with write capabilities.

decodeDuration

The encoded frame's decode duration.

Return Value

An error code. Returns `noErr` if there is no error.

Discussion

This function automatically sets the `kICMValidTime_DecodeDurationIsValid` flag.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMEncodedFrameSetDecodeTimeStamp

Sets an encoded frame's decode time stamp.

```
OSStatus ICMEncodedFrameSetDecodeTimeStamp (
 ICMMutableEncodedFrameRef   frame,
 TimeValue64                 decodeTimeStamp );
```

Parameters

frame

A reference to an encoded frame object with write capabilities.

decodeTimeStamp

The encoded frame's decode time stamp.

Return Value

An error code. Returns `noErr` if there is no error.

Discussion

This function automatically sets the `kICMValidTime_DecodeTimeStampIsValid` flag. If the display time stamp is valid, it also sets the `kICMValidTime_DisplayOffsetIsValid` flag.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMEncodedFrameSetDisplayDuration

Sets an encoded frame's display duration.

```
OSStatus ICMEncodedFrameSetDisplayDuration (
 ICMMutableEncodedFrameRef   frame,
 TimeValue64                 displayDuration );
```

Parameters

frame

A reference to an encoded frame object with write capabilities.

displayDuration

The encoded frame's display duration.

Return Value

An error code. Returns `noErr` if there is no error.

Discussion

This function automatically sets the `kICMValidTime_DisplayDurationIsValid` flag.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMEncodedFrameSetDisplayTimeStamp

Sets an encoded frame's display time stamp.

```
OSStatus ICMEncodedFrameSetDisplayTimeStamp (
 ICMMutableEncodedFrameRef   frame,
 TimeValue64                 displayTimeStamp );
```

Parameters

frame

A reference to an encoded frame object with write capabilities.

displayTimeStamp

The encoded frame's display time stamp.

Return Value

An error code. Returns `noErr` if there is no error.

Discussion

This function automatically sets the `kICMValidTime_DisplayTimeStampIsValid` flag. If the decode time stamp is valid, it also sets the `kICMValidTime_DisplayOffsetIsValid` flag.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMEncodedFrameSetValidTimeFlags

Sets an encoded frame's flags that indicate which of its time stamps and durations are valid.

```
OSStatus ICMEncodedFrameSetValidTimeFlags (
 ICMMutableEncodedFrameRef   frame,
 ICMValidTimeFlags           validTimeFlags );
```

Parameters

`frame`
A reference to an encoded frame object with write capabilities.

`validTimeFlags`
One of the following constants:

`kICMValidTime_DisplayTimeStampIsValid = 1L<<0`
The value of *`displayTimeStamp`* is valid.

`kICMValidTime_DisplayDurationIsValid = 1L<<1`
The value of *`displayDuration`* is valid.

Return Value
An error code. Returns `noErr` if there is no error.

Discussion
Setting an encoded frame's decode or display time stamp or duration automatically sets the corresponding valid time flags. For example, calling `ICMEncodedFrameSetDecodeTimeStamp` sets `kICMValidTime_DisplayTimeStampIsValid`. If both the encoded frame's decode time stamp and display time stamp are valid, `kICMValidTime_DisplayOffsetIsValid` is automatically set.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMEncodedFrameSetMediaSampleFlags

Sets the media sample flags for an encoded frame.

```
OSStatus ICMEncodedFrameSetMediaSampleFlags (
 ICMMutableEncodedFrameRef   frame,
 MediaSampleFlags            mediaSampleFlags );
```

Parameters

`frame`
A reference to an encoded frame object with write capabilities.

mediaSampleFlags

The object's media sample flags. These flags are listed in the header file `Movies.h`.

Return Value

An error code. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMEncodedFrameSetFrameType

Sets the `frame` type for an encoded frame.

```
OSStatus ICMEncodedFrameSetFrameType (
 ICMMutableEncodedFrameRef   frame,
 ICMFrameType                frameType );
```

Parameters

frame

A reference to an encoded frame object with write capabilities.

frameType

The frame type to be set:

`kICMFrameType_I = 'I'`

An I frame.

`kICMFrameType_P = 'P'`

A P frame.

`kICMFrameType_B = 'B'`

A B frame.

`kICMFrameType_Unknown = 0`

A frame of unknown type.

Return Value

An error code. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMEncodedFrameSetSimilarity

Sets the similarity for an encoded frame.

```
OSStatus ICMEncodedFrameSetSimilarity (
 ICMMutableEncodedFrameRef   frame,
 Float32                     similarity );
```

Parameters

frame

A reference to an encoded frame object with write capabilities.

similarity

The encoded frame's similarity value to be set. 1.0 means identical; 0.0 means not at all alike. The default value is –1.0, which means unknown.

Return Value

An error code. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMImageDescriptionGetProperty

Returns a particular property of a image description handle.

```
OSStatus ICMImageDescriptionGetProperty (
 ImageDescriptionHandle inDesc,
 ComponentPropertyClass inPropClass,
 ComponentPropertyID inPropID,
 ByteCount inPropValueSize,
 ComponentValuePtr outPropValueAddress,
 ByteCount *outPropValueSizeUsed );
```

Parameters

inDesc

The image description handle being interrogated.

inPropClass

The class of property being requested.

inPropID

The ID of the property being requested.

inPropValueSize

The size of the property value buffer.

outPropValueAddress

Points to the buffer to receive the property value.

outPropValueSizeUsed

Points to a variable to receive the actual size of returned property value. (This can be `NULL`).

Return Value
An error code. Returns `noErr` if there is no error.

Discussion
This routine returns a particular property of a image description handle.

Version Notes
Introduced in QuickTime 7

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMImageDescriptionGetPropertyInfo

Returns information about a particular property of a image description.

```
OSStatus ICMImageDescriptionGetPropertyInfo (
 ImageDescriptionHandle inDesc,
 ComponentPropertyClass inPropClass,
 ComponentPropertyID inPropID,
 ComponentValueType *outPropType,
 /*can be NULL */ ByteCount *outPropValueSize,
 /*can be NULL */ UInt32 *outPropertyFlags );
```

Parameters

inDesc
The image description handle being interrogated.

inPropClass
The class of property being requested.

inPropID
The ID of the property being requested.

outPropType
The type of property is returned here. (This can be `NULL`).

outPropValueSize
The size of property is returned here. (This can be `NULL`).

outPropertyFlags
The property flags are returned here. (This can be `NULL`).

Return Value
An error code. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 7

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMImageDescriptionSetProperty

Sets a particular property of a image description handle.

```
OSStatus ICMImageDescriptionSetProperty (
 ImageDescriptionHandle inDesc,
 ComponentPropertyClass inPropClass,
 ComponentPropertyID inPropID,
 ByteCount inPropValueSize,
 ConstComponentValuePtr inPropValueAddress );
```

Parameters

inDesc
: The image description handle being modified.

inPropClass
: The class of property being set.

inPropID
: The ID of the property being set.

inPropValueSize
: The size of property value.

inPropValueAddress
: Points to the property value buffer.

Return Value

An error code. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 7

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMMultiPassStorageCopyDataAtTimeStamp

Called by a multipass-capable compressor to retrieve data at a given time stamp.

```
OSStatus ICMMultiPassStorageCopyDataAtTimeStamp (
 ICMMultiPassStorageRef   multiPassStorage,
 TimeValue64              timeStamp,
 long                     index,
 CFMutableDataRef         *dataOut );
```

Parameters

multiPassStorage
: The multipass storage object.

timeStamp
: The time stamp at which the value should be retrieved.

index
: An index by which multiple values may be stored at a time stamp. The meaning of individual indexes is private to the compressor.

dataOut

A pointer to memory to receive the data at the time stamp.

Return Value

An error code. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMMultiPassStorageCreateWithCallbacks

Assembles a multipass storage mechanism from callbacks.

```
OSStatus ICMMultiPassStorageCreateWithCallbacks (
 CFAllocatorRef                  allocator,
 ICMMultiPassStorageCallbacks    *callbacks,
 ICMMultiPassStorageRef          *multiPassStorageOut );
```

Parameters

allocator

An allocator for this task. Pass `NULL` to use the default allocator.

callbacks

A structure containing a collection of callbacks for creating a custom multipass storage object. See `ICMMultiPassStorageCallbacks` (page 259).

multiPassStorageOut

A reference to the new multipass storage object.

Return Value

An error code. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMMultiPassStorageCreateWithTemporaryFile

Creates multipass storage using a temporary file.

```
OSStatus ICMMultiPassStorageCreateWithTemporaryFile (
 CFAllocatorRef                      allocator,
 FSRef                               *directoryRef,
 CFStringRef                         fileName,
 ICMMultiPassStorageCreationFlags    flags,
 ICMMultiPassStorageRef              *multiPassStorageOut );
```

Parameters

allocator

An allocator for this task. Pass `NULL` to use the default allocator.

directoryRef

A reference to a file directory. If you pass `NULL`, the ICM will use the user's Temporary Items folder.

fileName

A file name to use for the storage. If you pass `NULL`, the ICM will pick a unique name. If you pass the name of a file that already exists, the ICM will assume you are continuing a previous multipass session where you left off. This file will be deleted when the multipass storage is released, unless you set the `kICMMultiPassStorage_DoNotDeleteWhenDone` flag.

flags

Flag controlling this process:

`kICMMultiPassStorage_DoNotDeleteWhenDone = 1L<<0`

The temporary file should not be deleted when the multipass storage is released.

multiPassStorageOut

A reference to the new multipass storage.

Return Value

An error code. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMMultiPassStorageGetTimeStamp

Called by a multipass-capable compressor to retrieve a time stamp for which a value is stored.

```
OSStatus ICMMultiPassStorageGetTimeStamp (
 ICMMultiPassStorageRef    multiPassStorage,
 TimeValue64               fromTimeStamp,
 ICMMultiPassStorageStep   step,
 TimeValue64               *timeStampOut );
```

Parameters

multiPassStorage

The multipass storage object.

fromTimeStamp

The initial time stamp. This value is ignored for some values of *`step`*.

step
Indicates the kind of time stamp search to perform:

`kICMMultiPassStorage_GetFirstTimeStamp = 1`
Requests the first time stamp at which a value is stored.

`kICMMultiPassStorage_GetPreviousTimeStamp = 2`
Requests the previous time stamp before the time stamp specified in *fromTimeStamp* at which a value is stored.

`kICMMultiPassStorage_GetNextTimeStamp = 3`
Requests the next time stamp after the time stamp specified in *fromTimeStamp* at which a value is stored.

`kICMMultiPassStorage_GetLastTimeStamp = 4`
Requests the last time stamp at which a value is stored.

timeStampOut
A pointer to a `TimeValue64` value to receive the found time stamp. It will be set to –1 if no time stamp is found.

Return Value
An error code. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMMultiPassStorageGetTypeID

Returns the type ID for the current multipass storage object.

```
CFTypeID ICMMultiPassStorageGetTypeID ( void );
```

Return Value
A `CFTypeID` value.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

ICMMultiPassStorageRelease

Decrements the retain count of a multipass storage object.

```
void ICMMultiPassStorageRelease (
 ICMMultiPassStorageRef    multiPassStorage );
```

Parameters

multiPassStorageOut

A reference to a multipass storage object. You can create this object using `ICMMultiPassStorageCreateWithTemporaryFile` (page 173) or `ICMMultiPassStorageCreateWithCallbacks` (page 173). If you pass `NULL`, nothing happens.

Discussion

If the retain count drops to 0, the object is disposed.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMMultiPassStorageRetain

Increments the retain count of a multipass storage object.

```
ICMMultiPassStorageRef ICMMultiPassStorageRetain (
 ICMMultiPassStorageRef    multiPassStorage );
```

Parameters

multiPassStorageOut

A reference to a multipass storage object. You can create this object using `ICMMultiPassStorageCreateWithTemporaryFile` (page 173) or `ICMMultiPassStorageCreateWithCallbacks` (page 173). If you pass `NULL`, nothing happens.

Return Value

A reference to the object passed in *multiPassStorage*, for convenience.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ICMMultiPassStorageSetDataAtTimeStamp

Called by a multipass-capable compressor to store data at a given time stamp.

```
OSStatus ICMMultiPassStorageSetDataAtTimeStamp (
 ICMMultiPassStorageRef   multiPassStorage,
 TimeValue64              timeStamp,
 long                     index,
 CFDataRef                data );
```

Parameters

multiPassStorage

The multipass storage object.

timeStamp

The time stamp at which the value should be stored.

index

An index by which multiple values may be stored at a time stamp. The meaning of individual indexes is private to the compressor.

data

The data to be stored, or `NULL` to delete the value.

Return Value

An error code. Returns `noErr` if there is no error.

Discussion

The new data replaces any previous data held at that time stamp. If the value of *data* is `NULL`, the data for that time stamp is deleted. The format of the data is private to the compressor.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

ImageCodecBeginPass

Notifies the compressor that it should operate in multipass mode and use the given multipass storage.

```
ComponentResult ImageCodecBeginPass (
 ComponentInstance                ci,
 ICMCompressionPassModeFlags      passModeFlags,
 UInt32                           flags,
 ICMMultiPassStorageRef           multiPassStorage);
```

Parameters

ci

A component instance that identifies a connection to an image codec component.

passModeFlags

Indicates how the compressor should operate in this pass. If the `kICMCompressionPassMode_WriteToMultiPassStorage` flag is set, the compressor may gather information of interest and store it in `multiPassStorage`. If the `kICMCompressionPassMode_ReadFromMultiPassStorage` flag is set, the compressor may retrieve information from `multiPassStorage`. If the `kICMCompressionPassMode_OutputEncodedFrames flag` is set, the compressor must encode or drop every frame by calling `ICMCompressorSessionDropFrame` or `ICMCompressorSessionEmitEncodedFrame`. If that flag is not set, the compressor should not call these routines.

flags

Reserved. Ignore this parameter.

multiPassStorage

The multipass storage object that the compressor should use to store and retrieve information between passes.

Return Value

An error code, or `noErr` if there is no error.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `ImageCodec.h`

Declared In

`ImageCodec.h`

ImageCodecCompleteFrame

Directs the compressor to finish with a queued source frame, either emitting or dropping it.

```
ComponentResult ImageCodecCompleteFrame (
 ComponentInstance                ci,
 ICMCompressorSourceFrameRef      sourceFrame,
 UInt32                           flags );
```

Parameters

ci

A component instance that identifies a connection to an image codec component.

sourceFrame

The source frame that must be completed.

flags

Reserved; ignore.

Return Value

An error code, or `noErr` if there is no error.

Discussion

This frame does not necessarily need to be the first or only source frame emitted or dropped during this call, but the compressor must call either `ICMCompressorSessionDropFrame` or `ICMCompressorSessionEmitEncodedFrame` with this frame before returning. The ICM will call this function to force frames to be encoded for the following reasons: (a) the maximum frame delay count or maximum frame delay time in the `compressionSessionOptions` does not permit frames to be queued; (b) the client has called `ICMCompressionSessionCompleteFrames`.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Not supported C interface file: `ImageCodec.h`

Declared In
`ImageCodec.h`

ImageCodecDecodeBand

Returns an `ImageSubCodecDecompressRecord` structure for an image codec component.

```
ComponentResult ImageCodecDecodeBand (
 ComponentInstance                  ci,
 ImageSubCodecDecompressRecord      *drp,
 unsigned long                      flags );
```

Parameters

ci
: A component instance that identifies a connection to an image codec component.

drp
: A pointer to an `ImageSubCodecDecompressRecord` structure.

flags
: Not used; set to 0.

Return Value
An error code, or `noErr` if there is no error.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Not supported C interface file: `ImageCodec.h`

Declared In
`ImageCodec.h`

ImageCodecEncodeFrame

Presents the compressor with a frame to encode.

```
ComponentResult ImageCodecEncodeFrame (
 ComponentInstance                ci,
 ICMCompressorSourceFrameRef      sourceFrame,
 unsigned long                    flags );
```

Parameters

ci

A component instance that identifies a connection to an image codec component.

sourceFrame

The source frame to encode.

flags

Reserved; ignore.

Return Value

An error code, or `noErr` if there is no error.

Discussion

The compressor may encode the frame immediately or queue it for later encoding. If the compressor queues the frame for later decode, it must retain it (by calling `ICMCompressorSourceFrameRetain`) and release it when it is done with it (by calling `ICMCompressorSourceFrameRelease`). Pixel buffers are guaranteed to conform to the pixel buffer attributes returned by `ImageCodecPrepareToCompressFrames`. During multipass encoding, if the compressor requested the `kICMCompressionPassMode_NoSourceFrames` flag, the source frame pixel buffers may be `NULL`. (Note: this replaces `ImageCodecBandCompress`.)

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Not supported C interface file: `ImageCodec.h`

Declared In

`ImageCodec.h`

ImageCodecPrepareToCompressFrames

Prepares the compressor to receive frames.

```
ComponentResult ImageCodecPrepareToCompressFrames (
 ComponentInstance                ci,
 ICMCompressorSessionRef          session,
 ICMCompressionSessionOptionsRef  compressionSessionOptions,
 ImageDescriptionHandle           imageDescription,
 void                             *reserved,
 CFDictionaryRef                      *compressorPixelBufferAttributesOut);
```

Parameters

ci

A component instance that identifies a connection to an image codec component.

session

The compressor session reference. The compressor should store this in its globals; it will need it when calling the ICM back (for example, to call `ICMEncodedFrameCreateMutable` and `ICMCompressorSessionEmitEncodedFrame`). This is not a CF type. Do not call `CFRetain` or `CFRelease` on it.

compressionSessionOptions

The session options from the client. The compressor should retain this and use the settings to guide compression.

imageDescription

The image description. The compressor may add image description extensions.

reserved

Reserved for future use. Ignore this parameter.

compressorPixelBufferAttributesOut

The compressor should create a pixel buffer attributes dictionary and set `compressorPixelBufferAttributesOut` to it. The ICM will release it.

Return Value

An error code, or `noErr` if there is no error.

Discussion

The compressor should record session and retain `compressionSessionOptions` for use in later calls. The compressor may modify `imageDescription` at this point. The compressor should create and return pixel buffer attributes, which the ICM will release. (Note: this replaces `ImageCodecPreCompress`.)

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Not supported C interface file: `ImageCodec.h`

Declared In

`ImageCodec.h`

ImageCodecProcessBetweenPasses

Provides the compressor with an opportunity to perform processing between passes.

```
ComponentResult ImageCodecProcessBetweenPasses (
 ComponentInstance                ci,
 ICMMultiPassStorageRef           multiPassStorage,
 Boolean                          *interpassProcessingDoneOut,
 ICMCompressionPassModeFlags      *requestedNextPassModeFlagsOut );
```

Parameters

ci

A component instance that identifies a connection to an image codec component.

multiPassStorage

The multipass storage object that the compressor should use to store and retrieve information between passes.

interpassProcessingDoneOut

Points to a Boolean. Set this to `FALSE` if you want your `ImageCodecProcessBetweenPasses` function to be called again to perform more processing, `TRUE` if not.

requestedNextPassModeFlagsOut

Set `*requestedNextPassModeFlagsOut` to indicate the type of pass that should be performed next: To recommend a repeated analysis pass, set it to `kICMCompressionPassMode_ReadFromMultiPassStorage| kICMCompressionPassMode_WriteToMultiPassStorage`. To recommend a final encoding pass, set it to `kICMCompressionPassMode_ReadFromMultiPassStorage | kICMCompressionPassMode_OutputEncodedFrames`. If source frame buffers are not necessary for the recommended pass (for example, because all the required data has been copied into multipass storage), set `kICMCompressionPassMode_NoSourceFrames`.

Return Value

An error code, or `noErr` if there is no error.

Discussion

This function will be called repeatedly until it returns `TRUE` in `*interpassProcessingDoneOut`. The compressor may read and write to `multiPassStorage`. The compressor should indicate which type of pass it would prefer to perform next by setting `*requestedNextPassTypeOut`.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Not supported. C interface file: `ImageCodec.h`

Declared In

`ImageCodec.h`

InvokeQTTrackPropertyListenerUPP

Invokes the specified property listener of a track.

```
void InvokeQTTrackPropertyListenerUPP (
 Track inTrack,
 QTPropertyClass inPropClass,
 QTPropertyID inPropID,
 void *inUserData,
 QTTrackPropertyListenerUPP userUPP );
```

Parameters

inTrack

The track of this operation.

inPropClass

A property class.

inPropID

A property ID.

inUserData

A pointer to user data that will be passed to the callback.

userUPP

A `QTTrackPropertyListenerUPP` pointer.

Version Notes

Introduced in QuickTime 7

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

MediaContainsDisplayOffsets

Tests whether a media contains display offsets.

```
Boolean MediaContainsDisplayOffsets (
 Media    theMedia );
```

Parameters

theMedia
: The media for this operation. You obtain this media identifier from such functions as `NewTrackMedia` and `GetTrackMedia`.

Return Value
`TRUE` if the media is valid and contains at least one sample with a nonzero display offset; `FALSE` otherwise.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

MediaDecodeTimeToSampleNum

Finds the sample for a specified decode time.

```
void MediaDecodeTimeToSampleNum (
 Media          theMedia,
 TimeValue64    decodeTime,
 SInt64         *sampleNum,
 TimeValue64    *sampleDecodeTime,
 TimeValue64    *sampleDecodeDuration );
```

Parameters

theMedia
: The media for this operation. You obtain this media identifier from such functions as `NewTrackMedia` and `GetTrackMedia`.

decodeTime
: A 64-bit time value that represents the decode time for which you are retrieving sample information. You must specify this value in the media's time scale.

sampleNum
: A pointer to a variable that is to receive the sample number. The function returns the sample number that identifies the sample that contains data for the specified decode time, or 0 if it is not found.

sampleDecodeTime
A pointer to a time value. The function updates this time value to indicate the decode time of the sample specified by the *logicalSampleNum* parameter. This time value is expressed in the media's time scale. Set this parameter to `NULL` if you do not want this information.

sampleDecodeDuration
A pointer to a time value. The function updates this time value to indicate the decode duration of the sample specified by the *logicalSampleNum* parameter. This time value is expressed in the media's time scale. Set this parameter to `NULL` if you do not want this information.

Discussion
You can access this function's error returns through `GetMoviesError` and `GetMoviesStickyError`. It returns `paramErr` if there is a bad parameter value, `invalidTime` if *sampleDecodeTime* is out of the decode time range, or `noErr` if there is no error.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

MediaDisplayTimeToSampleNum

Fnds the sample number for a specified display time.

```
void MediaDisplayTimeToSampleNum (
 Media          theMedia,
 TimeValue64    displayTime,
 SInt64         *sampleNum,
 TimeValue64    *sampleDisplayTime,
 TimeValue64    *sampleDisplayDuration );
```

Parameters

theMedia
The media for this operation. You obtain this media identifier from such functions as `NewTrackMedia` and `GetTrackMedia`.

displayTime
A 64-bit time value that represents the display time for which you are retrieving sample information. You must specify this value in the media's time scale.

sampleNum
A pointer to a long integer that is to receive the sample number. The function returns the sample number that identifies the sample for the specified display time, or 0 if it is not found.

sampleDisplayTime
A pointer to a time value. The function updates this time value to indicate the display time of the sample specified by the *logicalSampleNum* parameter. This time value is expressed in the media's time scale. Set this parameter to `NULL` if you do not want this information.

sampleDisplayDuration
A pointer to a time value. The function updates this time value to indicate the display duration of the sample specified by the *logicalSampleNum* parameter. This time value is expressed in the media's time scale. Set this parameter to `NULL` if you do not want this information.

Discussion
You can access this function's error returns through `GetMoviesError` and `GetMoviesStickyError`. It returns `paramErr` if there is a bad parameter value, invalidTime if *`sampleDisplayTime`* is out of the display time range, or `noErr` if there is no error.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

MovieAudioExtractionBegin

Begins a movie audio extraction session.

```
OSStatus MovieAudioExtractionBegin (
 Movie                      m,
 UInt32                     flags,
 MovieAudioExtractionRef    *outSession );
```

Parameters

`m`

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromProperties`, `NewMovieFromFile`, and `NewMovieFromHandle`.

`flags`

Reserved; must be 0.

`outSession`

A pointer to an opaque session object.

Return Value
An error code. Returns `noErr` if there is no error.

Discussion
You must call this function before doing any movie audio extraction, because you will pass the object returned by *`outSession`* to the other movie audio extraction functions. The format of the extracted audio defaults to the summary channel layout of the movie (all right channels mixed together, all left surround channels mixed together, and so on.), 32-bit float, de-interleaved, with the sample rate set to the highest sample rate found in the movie. You can set the audio format to be something else, as long as it is uncompressed and you do it before your first call to `MovieAudioExtractionFillBuffer`.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

MovieAudioExtractionEnd

Ends a movie audio extraction session.

```
OSStatus MovieAudioExtractionEnd (
 MovieAudioExtractionRef    session );
```

Parameters

session

The session object returned by MovieAudioExtractionBegin (page 185).

Return Value

An error code. Returns `noErr` if there is no error.

Discussion

You must call this function when movie audio extraction is complete.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `Movies.h`

Declared In

`Movies.h`

MovieAudioExtractionFillBuffer

Extracts audio from a movie.

```
OSStatus MovieAudioExtractionFillBuffer (
 MovieAudioExtractionRef    session,
 UInt32                     *ioNumFrames,
 AudioBufferList            *ioData,
 UInt32                     *outFlags );
```

Parameters

session

The session object returned by MovieAudioExtractionBegin (page 185).

ioNumFrames

A pointer to the number of PCM frames to be extracted.

ioData

A pointer to an *AudioBufferList* allocated by the caller to hold the extracted audio data.

outFlags

A bit flag that indicates when extraction is complete:

`kMovieAudioExtractionComplete`

The extraction process is complete. Value is `(1L << 0)`.

Return Value

An error code. Returns `noErr` if there is no error.

Discussion

You call this function repeatedly; each call continues extracting audio where the last call left off. The function will extract as many of the requested PCM frames as it can, given the limits of the buffer supplied and the limits of the input movie. *ioNumFrames* will be updated with the exact number of valid frames being returned. When there is no more audio to extract from the movie, the function will continue to return `noErr` but will return no further audio data. In this case, the *outFlags* parameter will have its `kMovieAudioExtractionComplete` bit set. It is possible that the `kMovieAudioExtractionComplete` bit will accompany the last buffer of valid data.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `Movies.h`

Declared In

`Movies.h`

MovieAudioExtractionGetProperty

Gets a property of a movie audio extrraction session.

```
OSStatus MovieAudioExtractionGetProperty (
 MovieAudioExtractionRef    session,
 QTPropertyClass            inPropClass,
 QTPropertyID               inPropID,
 ByteCount                  inPropValueSize,
 QTPropertyValuePtr         outPropValueAddress,
 ByteCount                  *outPropValueSizeUsed);
```

Parameters

session

The session object returned by `MovieAudioExtractionBegin` (page 185).

inPropClass

Pass the following constant to define the property class: Property of an audio presentation; value is `'audi'`.

inPropID

Pass one of these constants to define the property ID:

`kAudioPropertyID_ChannelLayout`

The summary audio channel layout of a movie, or any other grouping of audio streams. All like-labeled channels are combined, without duplicates. For example, if there is a stereo (L/R) track, 5 single-channel tracks marked Left, Right, Left Surround, Right Surround and Center, and a 4-channel track marked L/R/Ls/Rs, then the summary AudioChannelLayout will be L/R/Ls/Rs/C, not L/R/L/R/Ls/Rs/C/L/R/Ls/Rs. The value of this constant is `'clay'`.

inPropValueSize

The size of the buffer allocated to receive the property value.

outPropValueAddress

A pointer to the buffer allocated to receive the property value.

outPropValueSizeUsed

The actual size of the property value.

Return Value

An error code. Returns `noErr` if there is no error.

Discussion

You can get and set more than just the channel layout. There are four properties, discussed below, all of which are gettable and settable (with some having restrictions on not setting after first calling `MovieAudioExtractionFillBuffer`).

Properties of the movie that is extracted from `kQTPropertyClass_MovieAudioExtraction_Movie` include the following movie class IDs:

- `kQTMovieAudioExtractionMoviePropertyID_CurrentTime`. The value is a `TimeRecord`, which you can set and get. When setting, you set the timescale to anything you want (for example, the output audio sample rate or the movie timescale). When getting, the timescale will be output audio sample rate for best accuracy.
- `kQTMovieAudioExtractionMoviePropertyID_AllChannelsDiscrete`. The value is Boolean (which is settable and gettable). Set to implement export of all audio channels without mixing. When this is set and the extraction audio stream basic description (ASBD) or channel layout are read back, you get information relating to the re-mapped movie.

Properties of the output audio extracted from `kQTPropertyClass_MovieAudioExtraction_Audio` include the following output audio class properties:

- `kQTMovieAudioExtractionAudioPropertyID_AudioStreamBasicDescription`. The value is an `AudioStreamBasicDescription`. You can get any time and set before first the `MovieAudioExtractionFillBuffer` call. If you get this property immediately after beginning an audio extraction session, it will tell you the default extraction format for the movie. This will include the number of channels in the default movie mix. If you set the output `AudioStreamBasicDescription`, it is recommended that you also set the output channel layout. If your output ASBD has a different number of channels than the default extraction mix, you must set the output channel layout. You can only set PCM output formats. Setting a compressed output format will fail.
- :`kQTMovieAudioExtractionAudioPropertyID_AudioChannelLayout`. The value is `AudioChannelLayout`, which you can get any time and set before first the `MovieAudioExtractionFillBuffer` call. If you get this property immediately after beginning an audio extraction session, it tells you what the channel layout is for the default extraction mix.

The information in this discussion also applies to the following functions:

- "MovieAudioExtractionGetPropertyInfo" (page 189)
- "MovieAudioExtractionSetProperty" (page 189)

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `Movies.h`

Declared In

`Movies.h`

MovieAudioExtractionGetPropertyInfo

Gets information about a property of a movie audio extraction session.

```
OSStatus MovieAudioExtractionGetPropertyInfo (
 MovieAudioExtractionRef    session,
 QTPropertyClass            inPropClass,
 QTPropertyID               inPropID,
 QTPropertyValueType        *outPropType,
 ByteCount                  *outPropValueSize,
 UInt32                     *outPropertyFlags );
```

Parameters

session

The session object returned by `MovieAudioExtractionBegin` (page 185).

inPropClass

Pass the following constant to define the property class: Property of an audio presentation; value is `'audi`

inPropID

Pass one of these constants to define the property ID:

`kAudioPropertyID_ChannelLayout`

The summary audio channel layout of a movie, or any other grouping of audio streams. All like-labeled channels are combined, without duplicates. For example, if there is a stereo (L/R) track, 5 single-channel tracks marked Left, Right, Left Surround, Right Surround and Center, and a 4-channel track marked L/R/Ls/Rs, then the summary AudioChannelLayout will be L/R/Ls/Rs/C, not L/R/L/R/Ls/Rs/C/L/R/Ls/Rs. The value of this constant is `'clay'`.

outPropType

A pointer to the type of the returned property's value.

outPropValueSize

A pointer to the size of the returned property's value.

outPropFlags

On return, a pointer to flags representing the requested information about the item's property.

Return Value

An error code. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `Movies.h`

Declared In

`Movies.h`

MovieAudioExtractionSetProperty

Sets a property of a movie audio extraction session.

```
OSStatus MovieAudioExtractionSetProperty (
 MovieAudioExtractionRef    session,
 QTPropertyClass            inPropClass,
 QTPropertyID               inPropID,
 ByteCount                  inPropValueSize,
 ConstQTPropertyValuePtr    inPropValueAddress );
```

Parameters

session

The session object returned by MovieAudioExtractionBegin (page 185).

inPropClass

Pass the following constant to define the property class: Property of an audio presentation; value is 'audi'.

inPropID

Pass one of these constants to define the property ID:

kAudioPropertyID_SummaryChannelLayout

The summary audio channel layout of a movie, or any other grouping of audio streams. All like-labeled channels are combined, without duplicates. For example, if there is a stereo (L/R) track, 5 single-channel tracks marked Left, Right, Left Surround, Right Surround and Center, and a 4-channel track marked L/R/Ls/Rs, then the summary AudioChannelLayout will be L/R/Ls/Rs/C, not L/R/L/R/Ls/Rs/C/L/R/Ls/Rs. The value of this constant is 'clay'.

inPropValueSize

The size of the property value.

inPropValueAddress

A *const void* pointer that points to the property value.

Return Value

An error code. Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: Movies.h

Declared In

Movies.h

NewMovieExportStageReachedCallbackUPP

Allocates a new Universal Procedure Pointer for a MovieExportStageReachedCallbackProc callback.

```
MovieExportStageReachedCallbackUPP NewMovieExportStageReachedCallbackUPP (
 MovieExportStageReachedCallbackProcPtr    userRoutine );
```

Parameters

userRoutine

A pointer to your application-defined callback function; see ICMDecompressionTrackingCallbackProc (page 254).

Return Value

A new Universal Procedure Pointer that you will use to invoke your callback.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `QuickTimeComponents.h`

Declared In
`QuickTimeComponents.h`

NewMovieFromProperties

Creates a new movie using movie properties.

```
OSStatus NewMovieFromProperties (
ItemCount                    inputPropertyCount,
QTNewMoviePropertyElement    *inputProperties,
ItemCount                    outputPropertyCount,
QTNewMoviePropertyElement    *outputProperties,
Movie                        *theMovie );
```

Parameters

`inputPropertyCount`
: The number of properties in the array passed in *`inputProperties`*.

`inputProperties`
: A pointer to a property array describing how to instantiate the movie. See `QTNewMoviePropertyElement` (page 260).

`outputPropertyCount`
: The number of properties in the array passed in *`outputProperties`*.

`outputProperties`
: A pointer to a property array to receive output parameters. See `QTNewMoviePropertyElement` (page 260). You may pass `NULL` if you don't want this information. The caller is responsible for calling the appropriate routines to dispose of any property values returned here. Since callers specify the property classes and IDs, they know who to call to dispose of the property values.

`theMovie`
: A pointer to a variable that receives the new movie.

Return Value
An error code. Returns `memFullErr` if the function could not allocate memory, `paramErr` if *`inputProperties`* or *`theMovie`* is `NULL`, or `noErr` if there is no error.

Discussion
This function can be used in all the cases where an existing `NewMovieFrom...` call is used. When calling this function, you supply a set of input properties that describe the information required to instantiate the movie (its data reference, audio context, visual context, and so on). You can also supply a set of output properties that you may be interested in; for example, information about whether the data reference was changed. See "New Movie Property Codes" (page 285).

This function verifies its input properties is as follows. First, the `propStatus` field of both the input and output property arrays is set to `kQTPropertyUnprocessedErr`. Then the input properties are checked one by one. If there is no problem with a property, its `propStatus` is set to `noErr` (0). If there is a problem, the `propStatus` for the property is set to 1 and the function returns `paramErr`. It is an error if a property is not recognized; `paramErr` is returned and the appropriate `propStatus` is set to `kQTPropertyNotSupportedErr`.

Another error is multiple data locations defined. In this case, the property status for the second data location is set to `paramErr`. It is not considered a fatal error if this function does not recognize an output property; the property's `propStatus` simply remains `kQTPropertyUnprocessedErr`.

The only output properties currently defined are those that support the behavior of functions of the form `NewMovieFrom....` For example, if you want to act upon the data reference being updated during the opening process, you would pass in the `kQTMovieInstantiationPropertyID_ResultDataLocationChanged` property.

This function must be used with `kQTContextPropertyID_VisualContext` to open a movie, for visual contexts to function with the movie. If you want to use visual contexts with a movie but want to inspect the movie prior to allocating the visual context to use (for instance you want to get the movie box), use `kQTContextPropertyID_VisualContext` with a `NULL` value. Otherwise, visual context calls with the movie will fail with an error. Using `GWorld` structures with the movie will also fail.

To handle special situations where this function cannot be used by your application, there is a method to switch a movie from GWorld mode to visual context mode. `SetMovieVisualContext` can be used to set a `NULL` visual context, which will disassociate the movie from its current visual context or GWorld. At this time, either `SetMovieGWorld` or `SetMovieVisualContext` can be used. If a movie is associated with a GWorld, visual context calls such as `GetMovieVisualContext` will fail. If a movie is a associated with a valid visual context, GWorld calls such as `GetMovieGWorld` will fail.

If a call to this function succeeds using a visual context or audio context, those objects will be explicitly retained for use by the movie. The movie object is responsible for releasing them. If you no longer need access to a context, it is safe to release it.

If no data location property is specified, then this function will behave like `NewMovie`, creating an empty movie. Thus `NewMovieFromProperties(0, nil, 0, nil, &movie)` is functionally equivalent to `movie = NewMovie(0)`.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

NewQTTrackPropertyListenerUPP

Creates a new callback to monitor a track property.

```
QTTrackPropertyListenerUPP NewQTTrackPropertyListenerUPP (
 QTTrackPropertyListenerProcPtr userRoutine );
```

Parameters

`userRoutine`

A pointer to a QTTrackPropertyListenerProcPtr callback.

Return Value
A new UPP; see Universal Procedure Pointers in the `QuickTime API Reference`.

Discussion
This routine creates a new callback to monitor a track property.

Version Notes
Introduced in QuickTime 7

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

QTAddTrackPropertyListener

Installs a callback to monitor a track property.

```
OSErr QTAddTrackPropertyListener (
 Track inTrack,
 QTPropertyClass inPropClass,
 QTPropertyID inPropID,
 QTTrackPropertyListenerUPP inListenerProc,
 void *inUserData );
```

Parameters

`inTrack`
The track for this operation.

`inPropClass`
A property class.

`inPropID`
A property ID.

`inListenerProc`
A Universal Procedure Pointer to a QTTrackPropertyListenerProc callback.

`inUserData`
A pointer to user data that will be passed to the callback.

Return Value
An error code. Returns `noErr` if there is no error.

Discussion
This routine installs a callback to monitor a track property.

Version Notes
Introduced in QuickTime 7

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

QTAudioContextCreateForAudioDevice

Creates a QTAudioContext object that encapsulates a connection to a CoreAudio output device.

```
OSStatus QTAudioContextCreateForAudioDevice (
 CFAllocatorRef allocator,
 CFStringRef coreAudioDeviceUID,
 CFDictionaryRef options,
 QTAudioContextRef *newAudioContextOut );
```

Parameters

allocator

Allocator used to create the audio context.

coreAudioDeviceUID

CoreAudio device UID. `NULL` means the default device.

options

Reserved. Pass `NULL`.

newAudioContextOut

Points to a variable to receive the new audio context.

Return Value

An error code. Returns `noErr` if there is no error.

Discussion

This routine creates a QTAudioContext object that encapsulates a connection to a CoreAudio output device. This object is suitable for passing to `SetMovieAudioContext` or `NewMovieFromProperties`, which targets the audio output of the movie to that device. A QTAudioContext object cannot be associated with more than one movie. Each movie needs its own connection to the device. In order to play more than one movie to a particular device, create a QTAudioContext object for each movie. You are responsible for releasing the QTAudioContext object created by this routine. After calling `SetMovieAudioContext` or `NewMovieFromProperties`, you can release the object since these APIs will retain it for their own use.

Version Notes

Introduced in QuickTime 7

Availability

Carbon status: Supported C interface file: `Movies.h`

Declared In

`Movies.h`

QTCopyMovieMetaData

Retains a movie's metadata object and returns it.

```
OSStatus QTCopyMovieMetaData (
Movie            inMovie,
QTMetaDataRef    *outMetaData );
```

Parameters

inMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromProperties`, `NewMovieFromFile`, and `NewMovieFromHandle`.

outMetaData

A pointer to an opaque metadata object wrapper associated with the movie passed in *inMovie*.

Return Value
Returns `invalidMovie` if the movie passed in *`inMovie`* is invalid, or `noErr` if there is no error.

Discussion
This function returns the metadata object associated with a movie. The object has retain/release semantics. It has already been retained before returning, but you should call `QTMetaDataRelease` when you are done. Because the movie can be disposed of at any time, the `QTMetaDataRef` may be valid when the movie no longer exists. In this case, the function will fail with a `kQTMetaDataInvalidMetaDataErr` error.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

QTCopyTrackMetaData

Retains a track's metadata object and returns it.

```
OSStatus QTCopyTrackMetaData (
Track            inTrack,
QTMetaDataRef    *outMetaData );
```

Parameters

`inTrack`
: A track identifier, which your application obtains from such functions as `NewMovieTrack` and `GetMovieTrack`.

`outMetaData`
: A pointer to an opaque metadata object wrapper associated with the track passed in *`inTrack`*.

Return Value
Returns `invalidMedia` if the track passed in *`inTrack`* is invalid, or `noErr` if there is no error.

Discussion
This function returns the metadata object associated with a track. The object has retain/release semantics. It has already been retained before returning, but you should call `QTMetaDataRelease` when you are done. Because the track can be disposed of at any time, the `QTMetaDataRef` may be valid when the track no longer exists. In this case, the function will fail with a `kQTMetaDataInvalidMetaDataErr` error.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

QTCopyMediaMetaData

Retains a media's metadata object and returns it.

```
OSStatus QTCopyMediaMetaData (
Media             inMedia,
QTMetaDataRef     *outMetaData );
```

Parameters

inMedia

The media for this operation. You obtain this media identifier from such functions as `NewTrackMedia` and `GetTrackMedia`.

outMetaData

A pointer to an opaque metadata object wrapper associated with the media passed in *inMedia*.

Return Value

Returns `invalidMedia` if the media passed in *inMedia* is invalid, or `noErr` if there is no error.

Discussion

This function returns the metadata object associated with a media. The object has retain/release semantics. It has already been retained before returning, but you should call `QTMetaDataRelease` when you are done. Because the media can be disposed of at any time, the `QTMetaDataRef` may be valid when the media no longer exists. In this case, the function will fail with a `kQTMetaDataInvalidMetaDataErr` error.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `Movies.h`

Declared In

`Movies.h`

QTGetTrackProperty

Returns the value of a specific track property.

```
OSErr QTGetTrackProperty (
 Track inTrack,
 QTPropertyClass inPropClass,
 QTPropertyID inPropID,
 ByteCount inPropValueSize,
 QTPropertyValuePtr outPropValueAddress,
 ByteCount *outPropValueSizeUsed );
```

Parameters

inTrack

The track for this operation.

inPropClass

A property class.

inPropID

A property ID.

inPropValueSize

The size of the buffer allocated to hold the property value.

outPropValueAddress

A pointer to the buffer allocated to hold the property value.

outPropValueSizeUsed

On return, the actual size of the value written to the buffer.

Return Value

An error code. Returns `noErr` if there is no error.

Discussion

This routine returns the value of a specific track property.

Version Notes

Introduced in QuickTime 7

Availability

Carbon status: Supported C interface file: `Movies.h`

Declared In

`Movies.h`

QTGetTrackPropertyInfo

Returns information about the properties of a track.

```
OSErr QTGetTrackPropertyInfo (
 Track inTrack,
 QTPropertyClass inPropClass,
 QTPropertyID inPropID,
 QTPropertyValueType *outPropType,
 ByteCount *outPropValueSize,
 UInt32 *outPropertyFlags );
```

Parameters

inTrack

The track for this operation.

inPropClass

A property class.

inPropID

A property ID.

outPropType

A pointer to memory allocated to hold the `property` type on return.

outPropValueSize

A pointer to memory allocated to hold the size of the property value on return.

outPropertyFlags

A pointer to memory allocated to hold property flags on return.

Return Value

An error code. Returns `noErr` if there is no error.

Discussion
This routine returns information about the properties of a track.

Version Notes
Introduced in QuickTime 7

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

QTMetaDataAddItem

Adds an inline metadata item to the metadata storage format.

```
OSStatus QTMetaDataAddItem (
QTMetaDataRef              inMetaData,
QTMetaDataStorageFormat    inMetaDataFormat,
QTMetaDataKeyFormat        inKeyFormat,
const UInt8                *inKeyPtr,
ByteCount                  inKeySize,
const UInt8                *inValuePtr,
ByteCount                  inValueSize,
UInt32                     inDataType,
QTMetaDataItem             *outItem);
```

Parameters

inMetaData
  The metadata object for this operation.

inMetaDataFormat
  The metadata storage format used by the object passed in *inMetaData*. The format may be `UserData` storage, iTunes metadata storage, or QuickTime metadata storage. Not all objects will include all forms of storage, and other storage formats may appear in the future. You cannot pass `kQTMetaDataStorageFormatWildcard` to target all storage formats.

inKeyFormat
  The format of the key.

inKeyPtr
  A pointer to the key of the item to be fetched next. You may pass `NULL` in this parameter if you are not interested in any specific key.

inKeySize
  The size of the key in bytes.

inValuePtr
  A pointer to the value to be added. This can be `NULL` if *inValueSize* is 0.

inValueSize
  The size of *inValuePtr* in bytes. Pass 0 if you want to add an item with no value.

inDataType

A data type from the following list:

```
kQTMetaDataTypeBinary              = 0,
kQTMetaDataTypeUTF8                = 1,
kQTMetaDataTypeUTF16BE             = 2,
kQTMetaDataTypeMacEncodedText      = 3,
kQTMetaDataTypeSignedIntegerBE     = 21,
kQTMetaDataTypeUnsignedIntegerBE   = 22,
kQTMetaDataTypeFloat32BE           = 23,
kQTMetaDataTypeFloat64BE           = 24
```

With `kQTMetaDataTypeSignedIntegerBE` and `kQTMetaDataTypeUnsignedIntegerBE`, the size of the integer is determined by the value size.

outItem

On return, a pointer to an opaque, unique `UInt64` identifier of the newly added item. Your application can use this to identify the metadata item within a metadata object for other metadata functions. You may pass `NULL` if you are not interested in the identifier of the newly added item. This identifier does not need to be disposed of.

Return Value

Returns `kQTMetaDataInvalidMetaDataErr` if the metadata object or its reference is invalid, `kQTMetaDataInvalidStorageFormatErr` if the metatada storage format is invalid, `kQTMetaDataInvalidKeyErr` if the key or its format is invalid, or `noErr` if there is no error. See "`Metadata Error Codes`" (page 285).

Discussion

The data type of the metadata item is assumed to be binary.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `Movies.h`

Declared In

`Movies.h`

QTMetaDataGetItemCount

Returns the number of items in a metadata storage format with a certain key.

```
OSStatus QTMetaDataGetItemCount (
QTMetaDataRef              inMetaData,
QTMetaDataStorageFormat    inMetaDataFormat,
QTMetaDataKeyFormat        inKeyFormat,
const UInt8                *inKeyPtr,
ByteCount                  inKeySize,
ItemCount                  *outCount);
```

Parameters

inMetaData

The metadata object for this operation.

inMetaDataFormat

The metadata storage format used by the object passed in *inMetaData*. The format may be `UserData` storage, iTunes metadata storage, or QuickTime metadata storage. Not all objects will include all forms of storage, and other storage formats may appear in the future. You cannot pass `kQTMetaDataStorageFormatWildcard` to target all storage formats.

inKeyFormat

The format of the key.

inKeyPtr

A pointer to the key of the item to be fetched next. You may pass `NULL` in this parameter if you are not interested in any specific key.

inKeySize

The size of the key in bytes.

outCount

The number of items in the metadata storage format that have the specified key.

Return Value

Returns `kQTMetaDataInvalidMetaDataErr` if the metadata object or its reference is invalid, `kQTMetaDataInvalidStorageFormatErr` if the metatada storage format is invalid, `kQTMetaDataInvalidKeyErr` if the key or its format is invalid, or `noErr` if there is no error. See "`Metadata Error Codes`" (page 285).

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `Movies.h`

QTMetaDataGetItemProperty

Returns a property of a metadata item.

```
OSStatus QTMetaDataGetItemProperty (
QTMetaDataRef          inMetaData,
QTMetaDataItem         inItem,
QTPropertyClass        inPropClass,
QTPropertyID           inPropID,
ByteCount              inPropValueSize,
QTPropertyValuePtr     outPropValueAddress,
ByteCount              *outPropValueSizeUsed );
```

Parameters

inMetaData

The metadata object for this operation.

inItem

The opaque, unique `UInt64` identifier of the metadata item for this operation. Your application obtains this item identifier from such functions as `QTMetaDataAddItem` (page 198) and `QTMetaDataGetNextItem` (page 203).

inPropClass

The class of the property being asked about.

inPropID

The ID of the property being asked about.

inPropValueSize

Size of the buffer allocated to receive the property value.

outPropValueAddress

A pointer to the buffer allocated to receive the item's property value.

outPropValueSizeUsed

On return, the actual size of buffer space used.

Return Value

Returns `kQTMetaDataInvalidMetaDataErr` if the metadata object or its reference is invalid, `kQTMetaDataInvalidItemErr` if the metatada item ID is invalid, `errPropNotSupported` if the metatada object does not support the property being asked about, `buffersTooSmall` if the allocated buffer is too small to hold the property, or `noErr` if there is no error. See "`Metadata Error Codes`" (page 285).

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `Movies.h`

Declared In

`Movies.h`

QTMetaDataGetItemPropertyInfo

Returns information about a property of a metadata item.

```
OSStatus QTMetaDataGetItemPropertyInfo (
QTMetaDataRef           inMetaData,
QTMetaDataItem          inItem,
QTPropertyClass         inPropClass,
QTPropertyID            inPropID,
QTPropertyValueType     *outPropType,
ByteCount               *outPropValueSize,
UInt32                  *outPropFlags );
```

Parameters

inMetaData

The metadata object for this operation.

inItem

The opaque, unique `UInt64` identifier of the metadata item for this operation. Your application obtains this item identifier from such functions as `QTMetaDataAddItem` (page 198) and `QTMetaDataGetNextItem` (page 203).

inPropClass

The class of the property being asked about.

inPropID

The ID of the property being asked about.

outPropType

A pointer to the type of the returned property's value.

outPropValueSize

A pointer to the size of the returned property's value.

outPropFlags
On return, a pointer to flags representing the requested information about the item's property.

Return Value
Returns `kQTMetaDataInvalidMetaDataErr` if the metadata object or its reference is invalid, `kQTMetaDataInvalidItemErr` if the metatada item ID is invalid, `errPropNotSupported` if the metatada object does not support the item property being asked about, or `noErr` if there is no error. See "`Metadata Error Codes`" (page 285).

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

QTMetaDataGetItemValue

Returns the value of a metadata item from an item identifier.

```
OSStatus QTMetaDataGetItemValue (
QTMetaDataRef      inMetaData,
QTMetaDataItem     inItem,
UInt8              *outValuePtr,
ByteCount          inValueSize,
ByteCount          *outActualSize );
```

Parameters

inMetaData
The metadata object for this operation.

inItem
The opaque, unique `UInt64` identifier of the metadata item for this operation. Your application can obtain this item identifier from such functions as `QTMetaDataAddItem` (page 198).

outValuePtr
A pointer to the first value of the item. You may pass `NULL` in this parameter if you just want to find out the size of the buffer needed.

inValueSize
The number of bytes in the *outValuePtr* buffer. You may pass 0 if you just want to find out the size of the buffer needed.

outActualSize
The actual size of the value if this parameter is not `NULL`.

Return Value
Returns `kQTMetaDataInvalidMetaDataErr` if the metadata object or its reference is invalid, `kQTMetaDataInvalidItemErr` if the metatada item ID is invalid, or `noErr` if there is no error. See "`Metadata Error Codes`" (page 285).

Discussion
You can use this function to get the value of a metadata item that has a known item identifier.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

QTMetaDataGetNextItem

Returns the next metadata item corresponding to a specified key.

```
OSStatus QTMetaDataGetNextItem (
QTMetaDataRef                inMetaData,
QTMetaDataStorageFormat      inMetaDataFormat,
QTMetaDataItem               inCurrentItem,
QTMetaDataKeyFormat          inKeyFormat,
const UInt8                  *inKeyPtr,
ByteCount                    inKeySize,
QTMetaDataItem               *outNextItem );
```

Parameters

inMetaData
The metadata object for this operation.

inMetaDataFormat
The metadata storage format used by the object passed in *inMetaData*. The format may be `UserData` storage, iTunes metadata storage, or QuickTime metadata storage. Not all objects will include all forms of storage, and other storage formats may appear in the future. Pass `kQTMetaDataStorageFormatWildcard` to target all storage formats.

inCurrentItem
The opaque, unique `UInt64` identifier of the current metadata item to start the search. Your application obtains this item identifier from such functions as `QTMetaDataAddItem` (page 198).

inKeyFormat
The format of the key.

inKeyPtr
A pointer to the key of the item to be fetched next. You may pass `NULL` in this parameter if you are not interested in any specific key.

inKeySize
The size of the key in bytes.

outNextItem
The ID of the next metadata item after the item specified by *inCurrentItem* that has the specified key.

Return Value
Returns `kQTMetaDataInvalidMetaDataErr` if the metadata object or its reference is invalid, `kQTMetaDataInvalidItemErr` if the metatada item ID is invalid, `kQTMetaDataInvalidStorageFormatErr` if the metatada storage format is invalid, `kQTMetaDataInvalidKeyErr` if the key or its format is invalid, `kQTMetaDataNoMoreItemErr` if the last item has been fetched, or `noErr` if there is no error. See "`Metadata Error Codes`" (page 285).

Discussion
If the item designated by *inCurrentItem* is kQTMetaDataItemUninitialized, the function returns the first item with the specified key in the storage format. If it refers to a valid item in the storage format, the function will return the next item with the key after the item designated by *inCurrentItem*.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: Movies.h

Declared In
Movies.h

QTMetaDataGetProperty

Returns a property of a metadata object.

```
OSStatus QTMetaDataGetProperty (
QTMetaDataRef          inMetaData,
QTPropertyClass        inPropClass,
QTPropertyID           inPropID,
ByteCount              inPropValueSize,
QTPropertyValuePtr     outPropValueAddress,
ByteCount              *outPropValueSizeUsed );
```

Parameters

inMetaData
The metadata object for this operation.

inPropClass
The class of the property being asked about.

inPropID
The ID of the property being asked about.

inPropValueSize
Size of the buffer allocated to receive the property value.

outPropValueAddress
A pointer to the buffer allocated to receive the property value.

outPropValueSizeUsed
On return, the actual size of buffer space used.

Return Value
Returns kQTMetaDataInvalidMetaDataErr if the metadata object or its reference is invalid, errPropNotSupported if the metatada object does not support the property being asked about, buffersTooSmall if the allocated buffer is too small to hold the property, or noErr if there is no error. See "Metadata Error Codes" (page 285).

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: Movies.h

Declared In
`Movies.h`

QTMetaDataGetPropertyInfo

Returns information about a property of a metadata object.

```
OSStatus QTMetaDataGetPropertyInfo (
QTMetaDataRef          inMetaData,
QTPropertyClass        inPropClass,
QTPropertyID           inPropID,
QTPropertyValueType    *outPropType,
ByteCount              *outPropValueSize,
UInt32                 *outPropFlags );
```

Parameters

`inMetaData`
The metadata object for this operation.

`inPropClass`
The class of the property being asked about.

`inPropID`
The ID of the property being asked about.

`outPropType`
A pointer to the type of the returned property's value.

`outPropValueSize`
A pointer to the size of the returned property's value.

`outPropFlags`
On return, a pointer to flags representing the requested information about the property.

Return Value
Returns `kQTMetaDataInvalidMetaDataErr` if the metadata object or its reference is invalid, `errPropNotSupported` if the metatada object does not support the property being asked about, or `noErr` if there is no error. See "`Metadata Error Codes`" (page 285).

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

QTMetaDataRetain

Increments the retain count of a metadata object.

```
QTMetaDataRef QTMetaDataRetain ( QTMetaDataRef inMetaData );
```

Parameters

inMetaData
A metadata object that you want to retain.

Return Value
If successful, returns a metadata object that is the same as that passed in *inMetaData*.

Discussion
This function retains a metadata object by incrementing its reference count. You should retain every metadata object when you receive it from elsewhere and you want it to persist. If you retain a metadata object you are responsible for releasing it by calling `QTMetaDataRelease` (page 206).

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

QTMetaDataRelease

Decrements the retain count of a metadata object.

```
void QTMetaDataRelease ( QTMetaDataRef inMetaData );
```

Discussion
This function releases a metadata object by decrementing its reference count. When the count becomes 0 the memory allocated to the object is freed and the object is destroyed. If you retain a metadata object you are responsible for releasing it when you no longer need it.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

QTMetaDataRemoveItem

Removes a metadata item from a storage format.

```
OSStatus QTMetaDataRemoveItem (
QTMetaDataRef     inMetaData,
QTMetaDataItem    inItem );
```

Parameters

inMetaData
The metadata object for this operation.

inItem

The opaque, unique `UInt64` identifier of the metadata item for this operation. Your application obtains this item identifier from such functions as `QTMetaDataAddItem` (page 198) and `QTMetaDataGetNextItem` (page 203).

Return Value

Returns `kQTMetaDataInvalidMetaDataErr` if the metadata object or its reference is invalid, `kQTMetaDataInvalidItemErr` if the metatada item ID is invalid, or `noErr` if there is no error. See "`Metadata Error Codes`" (page 285).

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `Movies.h`

Declared In

`Movies.h`

QTMetaDataRemoveItemsWithKey

Removes metadata items with a specific key from the storage format.

```
OSStatus QTMetaDataRemoveItemsWithKey (
QTMetaDataRef               inMetaData,
QTMetaDataStorageFormat     inMetaDataFormat,
QTMetaDataKeyFormat         inKeyFormat,
const UInt8                 *inKeyPtr,
ByteCount                   inKeySize);
```

Parameters

inMetaData

The metadata object for this operation.

inMetaDataFormat

The metadata storage format used by the object passed in *inMetaData*. The format may be `UserData` storage, iTunes metadata storage, or QuickTime metadata storage. Not all objects will include all forms of storage, and other storage formats may appear in the future. You can pass `kQTMetaDataStorageFormatWildcard` to target all storage formats.

inKeyFormat

The format of the key.

inKeyPtr

A pointer to the key of the item to be removed. You may pass `NULL` in this parameter if you want to remove all items.

inKeySize

The size of the key in bytes.

Return Value

Returns `kQTMetaDataInvalidMetaDataErr` if the metadata object or its reference is invalid, `kQTMetaDataInvalidStorageFormatErr` if the metatada storage format is invalid, `kQTMetaDataInvalidKeyErr` if the key or its format is invalid, or `noErr` if there is no error. See "`Metadata Error Codes`" (page 285).

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

QTMetaDataSetItem

Sets the value of the metadata item from the item identifier.

```
OSStatus QTMetaDataSetItem (
QTMetaDataRef      inMetaData,
QTMetaDataItem     inItem,
UInt8              *inValuePtr,
ByteCount          inValueSize,
UInt32             inDataType);
```

Parameters

inMetaData
The metadata object for this operation.

inItem
The opaque, unique `UInt64` identifier of the metadata item for this operation. Your application obtains this item identifier from such functions as `QTMetaDataAddItem` (page 198) and `QTMetaDataGetNextItem` (page 203).

inValuePtr
A pointer to the value to be set. This can be `NULL` if *inValueSize* is 0.

inValueSize
The size of *inValuePtr* in bytes. Pass 0 if you want to set an item with no value.

inDataType
A data type from the following list:

```
kQTMetaDataTypeBinary                = 0,
kQTMetaDataTypeUTF8                  = 1,
kQTMetaDataTypeUTF16BE               = 2,
kQTMetaDataTypeMacEncodedText        = 3,
kQTMetaDataTypeSignedIntegerBE       = 21,
kQTMetaDataTypeUnsignedIntegerBE     = 22,
kQTMetaDataTypeFloat32BE             = 23,
kQTMetaDataTypeFloat64BE             = 24
```

With kQTMetaDataTypeSignedIntegerBE and kQTMetaDataTypeUnsignedIntegerBE, the size of the integer is determined by the value size.

Return Value
Returns `kQTMetaDataInvalidMetaDataErr` if the metadata object or its reference is invalid, `kQTMetaDataInvalidItemErr` if the metatada item ID is invalid, or `noErr` if there is no error. See "`Metadata Error Codes`" (page 285).

Discussion
You can use this function to set the value of the metadata item with a given item identifier. You can set an item with an empty value by passing 0 in *inValueSize*.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

QTMetaDataSetItemProperty

Sets a property of a metadata item.

```
OSStatus QTMetaDataSetItemProperty (
QTMetaDataRef            inMetaData,
QTMetaDataItem           inItem,
QTPropertyClass          inPropClass,
QTPropertyID             inPropID,
ByteCount                inPropValueSize,
ConstQTPropertyValuePtr  inPropValueAddress );
```

Parameters

`inMetaData`
: The metadata object for this operation.

`inItem`
: The opaque, unique `UInt64` identifier of the metadata item for this operation. Your application obtains this item identifier from such functions as `QTMetaDataAddItem` (page 198) and `QTMetaDataGetNextItem` (page 203).

`inPropClass`
: The class of the property being set.

`inPropID`
: The ID of the property being set.

`inPropValueSize`
: Size of the buffer containing the property value being set.

`inPropValueAddress`
: A pointer to the buffer containing the item property value being set.

Return Value
Returns `kQTMetaDataInvalidMetaDataErr` if the metadata object or its reference is invalid, `kQTMetaDataInvalidItemErr` if the metatada item ID is invalid, `errPropNotSupported` if the metatada object does not support the property being set, `qtReadOnlyErr` if the property being set is read-only, or `noErr` if there is no error. See "`Metadata Error Codes`" (page 285).

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

QTMetaDataSetProperty

Sets a property of a metadata object.

```
OSStatus QTMetaDataSetProperty (
QTMetaDataRef              inMetaData,
QTPropertyClass            inPropClass,
QTPropertyID               inPropID,
ByteCount                  inPropValueSize,
ConstQTPropertyValuePtr    inPropValueAddress);
```

Parameters

`inMetaData`
The metadata object for this operation.

`inPropClass`
The class of the property being set.

`inPropID`
The ID of the property being set.

`inPropValueSize`
Size of the buffer containing the property value being set.

`inPropValueAddress`
A pointer to the buffer containing the property value being set.

Return Value

Returns `kQTMetaDataInvalidMetaDataErr` if the metadata object or its reference is invalid, `errPropNotSupported` if the metatada object does not support the property being set, `qtReadOnlyErr` if the property being set is read-only, or `noErr` if there is no error. See "`Metadata Error Codes`" (page 285).

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `Movies.h`

Declared In

`Movies.h`

QTOpenGLTextureContextCreate

Creates a new OpenGL texture context for a specified OpenGL context and pixel format.

```
OSStatus QTOpenGLTextureContextCreate (
 CFAllocatorRef                allocator,
 CGLContextObj                 cglContext,
 CGLPixelFormatObj             cglPixelFormat,
 CFDictionaryRef               attributes,
 QTOpenGLTextureContextRef     *newTextureContext );
```

Parameters

`allocator`
The allocator used to create the texture context.

cglContext
A pointer to an opaque `CGLPContextObj` structure representing the OpenGL context used to create textures. You can create this structure using `CGLCreateContext`.

cglPixelFormat
The pixel format object that specifies buffer types and other attributes of the new context.

attributes
A dictionary of attributes.

newTextureContext
A pointer to a variable to receive the new OpenGL texture context.

Return Value
An error code. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 7

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

QTPixelBufferContextCreate

Creates a new pixel buffer context with the given attributes.

```
OSStatus QTPixelBufferContextCreate (
 CFAllocatorRef allocator,
 CFDictionaryRef attributes,
 QTVisualContextRef *newPixelBufferContext );
```

Parameters

allocator
Allocator used to create the pixel buffer context.

attributes
Dictionary of attributes.

newPixelBufferContext
Points to a variable to receive the new pixel buffer context.

Return Value
An error code. Returns `noErr` if there is no error.

Discussion
This routine creates a new pixel buffer context with the given attributes.

Version Notes
Introduced in QuickTime 7

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

QTRemoveTrackPropertyListener

Removes a track property monitoring callback

```
OSErr QTRemoveTrackPropertyListener (
 Track inTrack,
 QTPropertyClass inPropClass,
 QTPropertyID inPropID,
 QTTrackPropertyListenerUPP inListenerProc,
 void *inUserData );
```

Parameters

inTrack
The track for this operation.

inPropClass
A property class.

inPropID
A property ID.

inListenerProc
A Universal Procedure Pointer to a QTTrackPropertyListenerProc callback.

inUserData
User data to be passed to the callback.

Return Value

An error code. Returns `noErr` if there is no error.

Discussion

This routine removes a track property monitoring callback.

Version Notes

Introduced in QuickTime 7

Availability

Carbon status: Supported C interface file: `Movies.h`

Declared In

`Movies.h`

QTSampleTableAddSampleDescription

Adds a sample description to a sample table, returning a sample description ID that can be used to refer to it.

```
OSStatus QTSampleTableAddSampleDescription (
 QTMutableSampleTableRef    sampleTable,
 SampleDescriptionHandle    sampleDescriptionH,
 long                       mediaSampleDescriptionIndex,
 QTSampleDescriptionID      *sampleDescriptionIDOut );
```

Parameters

sampleTable
A reference to an opaque sample table object.

sampleDescriptionH
: A handle to a `SampleDescription` structure. QuickTime will make its own copy of this handle.

mediaSampleDescriptionIndex
: The sample description index of this sample description in a media. Pass 0 for sample descriptions you add to sample tables, to indicate that this was not retrieved from a media.

sampleDescriptionIDOut
: A pointer to a variable to receive a sample description ID.

Return Value
An error code. Returns `noErr` if there is no error.

Discussion
You can use the returned sample description ID when adding samples to the sample table.

Special Considerations

Sample description IDs are local to each sample table. The same sample description handle may have different IDs when referenced in different sample tables.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

QTSampleTableAddSampleReferences

Adds sample references to a sample table.

```
OSStatus QTSampleTableAddSampleReferences (
 QTMutableSampleTableRef    sampleTable,
 SInt64                     dataOffset,
 ByteCount                  dataSizePerSample,
 TimeValue64                decodeDurationPerSample,
 TimeValue64                displayOffset,
 SInt64                     numberOfSamples,
 MediaSampleFlags           sampleFlags,
 QTSampleDescriptionID      sampleDescriptionID,
 SInt64                     *newSampleNumOut );
```

Parameters

sampleTable
: A reference to an opaque sample table object.

dataOffset
: A 64-bit signed integer that specifies the offset at which the first sample begins.

dataSizePerSample
: The number of bytes of data per sample. You must pass the data size per sample, not the total size of all the samples as with some other APIs.

decodeDurationPerSample
: A 64-bit time value that specifies the decode duration of each sample.

displayOffset

A 64-bit time value that specifies the offset from decode time to display time of each sample. If the decode times and display times are the same, pass 0.

numberOfSamples

A 64-bit signed integer, which must be greater than 0, that specifies the number of samples.

sampleFlags

Flags that indicate the `sync` status of all samples:

`mediaSampleNotSync`

If set to 1, indicates that the sample to be added is not a sync sample. Set this flag to 0 if the sample is a sync sample.

`mediaSampleShadowSync`

If set to 1, the sample is a shadow sync sample.

sampleDescriptionID

The ID of a sample description that has been added to the sample table with `QTSampleTableAddSampleDescription`.

newSampleNumOut

A 64-bit signed integer that points to a variable to receive the sample number of the first sample that was added. Pass `NULL` if you don't want this information.

Return Value

An error code. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `Movies.h`

Declared In

`Movies.h`

QTSampleTableCopySampleDescription

Retrieves a sample description from a sample table.

```
OSStatus QTSampleTableCopySampleDescription (
 QTSampleTableRef            sampleTable,
 QTSampleDescriptionID       sampleDescriptionID,
 long                        *mediaSampleDescriptionIndexOut,
 SampleDescriptionHandle     *sampleDescriptionHOut );
```

Parameters

sampleTable

A reference to an opaque sample table object.

sampleDescriptionID

The sample description ID.

mediaSampleDescriptionIndexOut

A pointer to a variable to receive a media sample description index. If the sample description came from a media, this is the index that could be passed to `GetMediaSampleDescription` to retrieve the same sample description handle. The index will be 0 if the sample description did not come directly from a media. Pass `NULL` if you do not want to receive this information.

sampleDescriptionHOut

A pointer to a variable to receive a newly allocated sample description handle. Pass `NULL` if you do not want one. The caller is responsible for disposing the returned sample description handle using `DisposeHandle`.

Return Value

An error code. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `Movies.h`

Declared In

`Movies.h`

QTSampleTableCreateMutable

Creates a new, empty sample table.

```
OSStatus QTSampleTableCreateMutable (
 CFAllocatorRef              allocator,
 TimeScale                   timescale,
 void                        *hints,
 QTMutableSampleTableRef     *newSampleTable );
```

Parameters

allocator

The allocator to use for the new sample table.

timescale

A long integer that represents the timescale to use for durations and display offsets.

hints

Reserved; pass `NULL`.

newSampleTable

A pointer to a variable that receives a new reference to an opaque sample table object.

Return Value

An error code. Returns `memFullErr` if it could not allocate memory, `paramErr` if the time scale is not positive or *newSampleTable* is `NULL`, or `noErr` if there is no error.

Discussion

The newly created sample table contains no sample references. When sample references are added, their durations and display offsets are interpreted according to the sample table's current timescale.

Version Notes

Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

QTSampleTableCreateMutableCopy

Copies a sample table.

```
OSStatus QTSampleTableCreateMutableCopy (
 CFAllocatorRef             allocator,
 QTSampleTableRef           sampleTable,
 void                       *hints,
 QTMutableSampleTableRef    *newSampleTable );
```

Parameters

`allocator`
: The allocator to use for the new sample table.

`sampleTable`
: A reference to an opaque sample table object to copy.

`hints`
: Reserved; set to `NULL`.

`newSampleTable`
: A pointer to a variable that receives a reference to an opaque sample table object.

Return Value
An error code. Returns `memFullErr` if it could not allocate memory, `paramErr` if the time scale is not positive or *`newSampleTable`* is `NULL`, or `noErr` if there is no error.

Discussion
All the sample references and sample descriptions in the sample table are copied.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

QTSampleTableGetDataOffset

Returns the data offset of a sample.

```
SInt64 QTSampleTableGetDataOffset (
 QTSampleTableRef    sampleTable,
 SInt64              sampleNum );
```

Parameters

sampleTable

A reference to an opaque sample table object.

sampleNum

A 64-bit signed integer that represents a sample number. The first sample's number is 1.

Return Value

A 64-bit signed integer that represents the offset to the sample. Returns 0 if *sampleTable* is `NULL` or if the sample number is out of range.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `Movies.h`

Declared In

`Movies.h`

QTSampleTableGetDataSizePerSample

Returns the data size of a sample.

```
ByteCount QTSampleTableGetDataSizePerSample (
 QTSampleTableRef    sampleTable,
 SInt64              sampleNum );
```

Parameters

sampleTable

A reference to an opaque sample table object.

sampleNum

A 64-bit signed integer that represents the sample number. The first sample's number is 1.

Return Value

The size of the sample in bytes. Returns 0 if *samplTable* is `NULL` or if the sample number is out of range.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `Movies.h`

Declared In

`Movies.h`

QTSampleTableGetDecodeDuration

Returns the decode duration of a sample.

```
TimeValue64 QTSampleTableGetDecodeDuration (
 QTSampleTableRef    sampleTable,
 SInt64              sampleNum );
```

Parameters

sampleTable

A reference to an opaque sample table object.

sampleNum

A 64-bit signed integer that represents the sample number. The first sample's number is 1.

Return Value
A 64-bit time value that represents the decode duration of the sample. Returns 0 if *samplTable* is `NULL` or if the sample number is out of range.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

QTSampleTableGetDisplayOffset

Returns the offset from decode time to display time of a sample.

```
TimeValue64 QTSampleTableGetDisplayOffset (
 QTSampleTableRef    sampleTable,
 SInt64              sampleNum );
```

Parameters

sampleTable

A reference to an opaque sample table object.

sampleNum

A 64-bit signed integer that represents the sample number. The first sample's number is 1.

Return Value
A 64-bit time value that represents the offset from decode time to display time of the sample. Returns 0 if *samplTable* is `NULL` or if the sample number is out of range.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

QTSampleTableGetNextAttributeChange

Finds the next sample number at which one or more of a set of given sample attributes change.

```
OSStatus QTSampleTableGetNextAttributeChange (
 QTSampleTableRef          sampleTable,
 SInt64                    startSampleNum,
 QTSampleTableAttribute    attributeMask,
 SInt64                    *sampleNumOut );
```

Parameters

sampleTable

A reference to an opaque sample table object.

startSampleNum

A 64-bit signed integer that contains the sample number to start searching from.

attributeMask

An unsigned 32-bit integer that contains flags indicating which kinds of attribute changes to search for:

`kQTSampleTableAttribute_DiscontiguousData = 1L << 0`

Set this flag to find the first sample number *num* such that samples *num-1* and *num* are not adjacent; that is, `dataOffset` of `num-1 + dataSize` of `num-1 != dataOffset` of `num`.

`kQTSampleTableAttribute_DataSizePerSampleChange = 1L << 1`

Set this flag to find the first sample with data size per sample different from that of the starting sample.

`kQTSampleTableAttribute_DecodeDurationChange = 1L << 2`

Set this flag to find the first sample with decode duration different from that of the starting sample.

`kQTSampleTableAttribute_DisplayOffsetChange = 1L << 3`

Set this flag to find the first sample with display offset different from that of the starting sample.

`kQTSampleTableAttribute_SampleDescriptionIDChange = 1L << 4`

Set this flag to find the first sample with sample description ID different from that of the starting sample.

`kQTSampleTableAttribute_SampleFlagsChange = 1L << 5`

Set this flag to find the first sample with any media sample flags different from those of the starting sample.

`kQTSampleTableAnyAttributeChange = 0`

If no flags are set, find the first sample with any attribute different from the starting sample.

sampleNumOut

A 64-bit signed integer that points to a variable to receive the next sample number after *startSampleNum* at which any of the requested attributes change. If no attribute changes are found, this variable is set to 0.

Return Value

An error code. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

QTSampleTableGetNumberOfSamples

Returns the number of samples in a sample table.

```
SInt64 QTSampleTableGetNumberOfSamples (
 QTSampleTableRef    sampleTable );
```

Parameters

`sampleTable`

A reference to an opaque sample table object.

Return Value
A 64-bit signed integer that contains the number of samples, or 0 if *`sampleTable`* is `NULL`.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

QTSampleTableGetProperty

Returns the value of a specific sample table property.

```
OSStatus QTSampleTableGetProperty (
 QTSampleTableRef      sampleTable,
 QTPropertyClass       inPropClass,
 QTPropertyID          inPropID,
 ByteCount             inPropValueSize,
 QTPropertyValuePtr    outPropValueAddress,
 ByteCount             *outPropValueSizeUsed );
```

Parameters

`sampleTable`

A reference to an opaque sample table object.

`inPropClass`

Pass the following constant to define the property class:

`kQTPropertyClass_SampleTable = 'qtst'`

Property of a sample table.

inPropID
Pass one of these constants to define the property ID:

`kQTSampleTablePropertyID_TotalDecodeDuration = 'tded'`
The total decode duration of all samples in the sample table. Read-only.

`kQTSampleTablePropertyID_MinDisplayOffset = '<ddd'`
The least display offset in the table. Negative offsets are less than positive offsets. Read-only.

`kQTSampleTablePropertyID_MaxDisplayOffset = '>ddd'`
The greatest display offset in the table. Positive offsets are greater than negative offsets. Read-only.

`kQTSampleTablePropertyID_MinRelativeDisplayTime = '<dis'`
The least display time of all samples in the table, relative to the decode time of the first sample in the table. Read-only.

`kQTSampleTablePropertyID_MaxRelativeDisplayTime = '>dis'`
The greatest display time of all samples in the table, relative to the decode time of the first sample in the table. Read-only.

inPropValueSize
The size of the buffer allocated to receive the property value.

outPropValueAddress
A pointer to the buffer allocated to receive the property value.

outPropValueSizeUsed
On return, the actual size of the property value written to the buffer.

Return Value
An error code. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

QTSampleTableGetPropertyInfo

Returns information about the properties of a sample table.

```
OSStatus QTSampleTableGetPropertyInfo (
 QTSampleTableRef       sampleTable,
 QTPropertyClass        inPropClass,
 QTPropertyID           inPropID,
 QTPropertyValueType    *outPropType,
 ByteCount              *outPropValueSize,
 UInt32                 *outPropertyFlags );
```

Parameters

sampleTable

A reference to an opaque sample table object.

inPropClass

Pass the following constant to define the property class:

`kQTPropertyClass_SampleTable = 'qtst'`

Property of a sample table.

inPropID

Pass one of these constants to define the property ID:

`kQTSampleTablePropertyID_TotalDecodeDuration = 'tded'`

The total decode duration of all samples in the sample table. Read-only.

`kQTSampleTablePropertyID_MinDisplayOffset = '<ddd'`

The least display offset in the table. Negative offsets are less than positive offsets. Read-only.

`kQTSampleTablePropertyID_MaxDisplayOffset = '>ddd'`

The greatest display offset in the table. Positive offsets are greater than negative offsets. Read-only.

`kQTSampleTablePropertyID_MinRelativeDisplayTime = '<dis'`

The least display time of all samples in the table, relative to the decode time of the first sample in the table. Read-only.

`kQTSampleTablePropertyID_MaxRelativeDisplayTime = '>dis'`

The greatest display time of all samples in the table, relative to the decode time of the first sample in the table. Read-only.

outPropType

A pointer to memory allocated to hold the property type on return: Pass `NULL` if you do not want this information.

outPropValueSize

A pointer to memory allocated to hold the size of the property value on return. Pass `NULL` if you do not want this information.

outPropertyFlags

A pointer to memory allocated to hold property flags on return. Pass `NULL` if you do not want this information.

Return Value

An error code. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

QTSampleTableGetSampleDescriptionID

Returns the sample description ID of a sample.

```
QTSampleDescriptionID QTSampleTableGetSampleDescriptionID (
 QTSampleTableRef    sampleTable,
 SInt64              sampleNum );
```

Parameters

sampleTable

A reference to an opaque sample table object.

sampleNum

A 64-bit signed integer that represents the sample number. The first sample's number is 1.

Return Value
The sample's sample description ID. Returns 0 if *samplTable* is `NULL` or if the sample number is out of range.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

QTSampleTableGetSampleFlags

Returns the media sample flags of a sample.

```
MediaSampleFlags QTSampleTableGetSampleFlags (
 QTSampleTableRef    sampleTable,
 SInt64              sampleNum );
```

Parameters

sampleTable

A reference to an opaque sample table object.

sampleNum

A 64-bit signed integer that represents the sample number. The first sample's number is 1.

Return Value
A constant that describes characteristics of the sample (see below). Returns 0 if *samplTable* is `NULL` or if the sample number is out of range.

Discussion
This function can return one or more of the following constants:

`mediaSampleNotSync`
: Sample is not a sync sample (for example, it is is frame differenced).

`mediaSampleShadowSync`
: Sample is a shadow sync sample.

`mediaSampleDroppable`
: Sample does not need to be decoded for later samples to be decoded properly.

`mediaSamplePartialSync`
: Sample is a partial sync sample (for example, 1 frame after open GOP).

`mediaSampleHasRedundantCoding`
: Sample is known to contain redundant coding.

`mediaSampleHasNoRedundantCoding`
: Sample is known not to contain redundant coding.

`mediaSampleIsDependedOnByOthers`
: One or more other samples depend on this sample being decoded.

`mediaSampleIsNotDependedOnByOthers`
: Synonym for `mediaSampleDroppable`.

`mediaSampleDependsOnOthers`
: Decoding this sample depends on decoding other samples.

`mediaSampleDoesNotDependOnOthers`
: Decoding this sample does not depend on decoding other samples.

`mediaSampleEarlierDisplayTimesAllowed`
: Samples later in decode order may have earlier display times.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

QTSampleTableGetTimeScale

Returns the timescale of a sample table.

```
TimeScale QTSampleTableGetTimeScale (
 QTSampleTableRef    sampleTable );
```

Parameters

`sampleTable`
: A reference to an opaque sample table object.

Return Value
A long integer that represents the sample's time scale, or 0 if *`sampleTable`* is `NULL`.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

QTSampleTableGetTypeID

Returns the `CFTypeID` value for the current sample table.

```
CFTypeID QTSampleTableGetTypeID ( void );
```

Return Value
A `CFTypeID` value.

Discussion
You could use this to test whether a `CFTypeRef` that was extracted from a CF container such as a `CFArray` is a `QTSampleTableRef`.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

QTSampleTableRelease

Decrements the retain count of a sample table.

```
void QTSampleTableRelease (
 QTSampleTableRef    sampleTable );
```

Parameters

`sampleTable`
A reference to an opaque sample table object. If you pass `NULL` in this parameter, nothing happens.

Discussion
If the retain count decreases to zero, the sample table is disposed.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

QTSampleTableReplaceRange

Replaces a range of samples in a sample table with a range of samples from another sample table.

```
OSStatus QTSampleTableReplaceRange (
 QTMutableSampleTableRef    destSampleTable,
 SInt64                     destStartingSampleNum,
 SInt64                     destSampleCount,
 QTSampleTableRef           sourceSampleTable,
 SInt64                     sourceStartingSampleNum,
 SInt64                     sourceSampleCount );
```

Parameters

destSampleTable
A reference to an opaque sample table object to be modified.

destStartingSampleNum
A 64-bit signed integer that represents the first sample number in *destSampleTable* to be replaced or deleted, or the sample number at which samples should be inserted.

destSampleCount
A 64-bit signed integer that represents the number of samples to be removed from *destSampleTable*. Pass 0 to insert samples without removing samples.

sourceSampleTable
A reference to an opaque sample table object from which samples should be copied, or `NULL` to delete samples.

sourceStartingSampleNum
A 64-bit signed integer that represents the first sample number to be copied. This parameter is ignored when deleting samples.

sourceSampleCount
A 64-bit signed integer that represents the number of samples which should be copied. Pass 0 to delete samples.

Return Value
An error code. Returns `noErr` if there is no error.

Discussion
This function removes *destSampleCount* samples from *destSampleTable* starting with *destStartingSampleNum*, and then inserts *sourceSampleCount* samples from *sourceSampleTable* starting with *sourceStartingSampleNum* where the removed samples were. Sample descriptions will be copied if necessary and new sample description IDs defined. This function can also be used to delete a range of samples, or to insert samples without removing any.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

QTSampleTableRetain

Increments the retain count of a sample table.

```
QTSampleTableRef QTSampleTableRetain (
 QTSampleTableRef    sampleTable );
```

Parameters

sampleTable

A reference to an opaque sample table object. If you pass `NULL` in this parameter, nothing happens.

Return Value

A pointer to the `OpaqueQTSampleTable` structure that is returned for your convenience, or `NULL` if the function fails.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `Movies.h`

Declared In

`Movies.h`

QTSampleTableSetProperty

Sets the value of a specific sample table property.

```
OSStatus QTSampleTableSetProperty (
 QTSampleTableRef           sampleTable,
 QTPropertyClass            inPropClass,
 QTPropertyID               inPropID,
 ByteCount                  inPropValueSize,
 ConstQTPropertyValuePtr    inPropValueAddress );
```

Parameters

sampleTable

A reference to an opaque sample table object.

inPropClass

Pass the following constant to define the property class:

`kQTPropertyClass_SampleTable = 'qtst'`

Property of a sample table.

inPropID

Pass one of these constants to define the property ID:

`kQTSampleTablePropertyID_TotalDecodeDuration = 'tded'`

The total decode duration of all samples in the sample table. Read-only.

`kQTSampleTablePropertyID_MinDisplayOffset = '<ddd'`

The least display offset in the table. Negative offsets are less than positive offsets. Read-only.

`kQTSampleTablePropertyID_MaxDisplayOffset = '>ddd'`

The greatest display offset in the table. Positive offsets are greater than negative offsets. Read-only.

`kQTSampleTablePropertyID_MinRelativeDisplayTime = '<dis'`

The least display time of all samples in the table, relative to the decode time of the first sample in the table. Read-only.

`kQTSampleTablePropertyID_MaxRelativeDisplayTime = '>dis'`

The greatest display time of all samples in the table, relative to the decode time of the first sample in the table. Read-only.

inPropValueSize

Pass the size of the property value.

inPropValueAddress

Pass a *const void* pointer to the property value.

Return Value
An error code. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

QTSampleTableSetTimeScale

Changes the timescale of a sample table.

```
OSStatus QTSampleTableSetTimeScale (
 QTMutableSampleTableRef    sampleTable,
 TimeScale                  newTimeScale );
```

Parameters

sampleTable

A reference to an opaque sample table object.

newTimeScale

A long integer whose value is the time scale to be set.

Return Value
An error code. Returns `paramErr` if the time scale is not positive or *sampleTable* is `NULL`, or `noErr` if there is no error.

Discussion
The durations and display offsets of all the sample references in the sample table are scaled from the old timescale to the new timescale. No durations are scaled to a value less than 1. Display offsets are adjusted to avoid display time collisions.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

QTSetTrackProperty

Sets the value of a specific track property.

```
OSErr QTSetTrackProperty (
 Track inTrack,
 QTPropertyClass inPropClass,
 QTPropertyID inPropID,
 ByteCount inPropValueSize,
 ConstQTPropertyValuePtr inPropValueAddress );
```

Parameters

inTrack
The track for this operation.

inPropClass
A property class.

inPropID
A property ID.

inPropValueSize
The size of the property value.

inPropValueAddress
A pointer to the the property value.

Return Value
An error code. Returns `noErr` if there is no error.

Discussion
This routine sets the value of a specific track property.

Version Notes
Introduced in QuickTime 7

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

QTSoundDescriptionConvert

Converts a sound description from one version to another.

```
OSStatus QTSoundDescriptionConvert (
QTSoundDescriptionKind    fromKind,
SoundDescriptionHandle    fromDescription,
QTSoundDescriptionKind    toKind,
SoundDescriptionHandle    *toDescription );
```

Parameters

fromKind

Reserved. Set to `kSoundDescriptionKind_Movie_AnyVersion`.

fromDescription

A handle to the sound description to be converted.

toKind

The version you want *fromDescription* to be.

toDescription

A reference to the resulting `SoundDescription` structure. You must dispose of the reference using `DisposeHandle`.

Return Value

An error code. Returns `noErr` if there is no error.

Discussion

The *fromKind* parameter is reserved for future expansion; at present you must set it to `kSoundDescriptionKind_Movie_AnyVersion`. Depending on the value you pass in *toKind*, you can specify that you would like a specific `SoundDescription` version, the lowest possible version (given the constraints of the format described by *fromDescription*), or any version at all.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `Movies.h`

Declared In

`Movies.h`

QTSoundDescriptionCreate

Creates a sound description structure of the requested kind from an `AudioStreamBasicDescription`, optional audio channel layout, and optional magic cookie.

```
OSStatus QTSoundDescriptionCreate (
AudioStreamBasicDescription    *inASBD,
AudioChannelLayout             *inLayout,
ByteCount                      inLayoutSize,
void                           *inMagicCookie
ByteCount                      inMagicCookieSize
QTSoundDescriptionKind         inRequestedKind
SoundDescriptionHandle         *outSoundDesc );
```

Parameters

inASBD

A description of the format.

inLayout

The audio channel layout (can be `NULL` if there isn't one).

inLayoutSize

The size of the audio channel layout (should be 0 if *inLayout* is `NULL`).

inMagicCookie

The magic cookie for the decompressor (can be `NULL` if the decompressor doesn't require one).

inMagicCookieSize

The size of the magic cookie (should be 0 if the *inMagicCookie* parameter is `NULL`).

inRequestedKind

The kind of sound description to create.

outSoundDesc

The resulting sound description. The caller must dispose of it with `DisposeHandle`.

Return Value
An error code. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

QTSoundDescriptionGetPropertyInfo

Gets information about a particular property of a sound description.

```
OSStatus QTSoundDescriptionGetPropertyInfo (
SoundDescriptionHandle     inDesc,
QTPropertyClass            inPropClass,
QTPropertyID               inPropID,
QTPropertyValueType        *outPropType,
ByteCount                  *outPropValueSize,
UInt32                      *outPropertyFlags);
```

Parameters

inDesc

The sound description being interrogated.

inPropClass
: The class of the property being requested.

inPropID
: The ID of the property being requested.

outPropType
: The type of the property returned here (can be `NULL`).

outPropValueSize
: The size of the property returned here (can be `NULL`).

outPropertyFlags
: The property flags returned here (can be `NULL`).

Return Value
An error code. Returns `noErr` if there is no error.

Discussion
The following constants identify sound description properties.

```
enum {
    kQTSoundDescriptionPropertyID_AudioChannelLayout = 'clay',
    kQTSoundDescriptionPropertyID_MagicCookie = 'kuki',
    kQTSoundDescriptionPropertyID_AudioStreamBasicDescription = 'asbd',
    kQTSoundDescriptionPropertyID_UserReadableText = 'text'
};
```

Special Considerations

`kQTSoundDescriptionPropertyID_AudioChannelLayout = 'clay'`
: Used to get or set an `AudioChannelLayout` value. This is a variable-size property because it may contain an array of Channel Descriptions. You must get the size by calling `QTSoundDescriptionGetPropertyInfo`, allocate a structure of that size, and then get the property.

`kQTSoundDescriptionPropertyID_MagicCookie = 'kuki'`
: Used to get or set opaque bytes. This is a variable-size property, because it is completely defined by the codec that uses the cookie. You must get the size by calling `QTSoundDescriptionGetPropertyInfo`, allocate a structure of that size, and then get the property.

`kQTSoundDescriptionPropertyID_AudioStreamBasicDescription = 'asbd'`
: Used to get an `AudioStreamBasicDescription` value.

`kQTSoundDescriptionPropertyID_UserReadableText = 'text'`
: Used to get a `CFStringRef` value. `QTSoundDescriptionGetProperty` does a `CFRetain` of the returned `CFString` on behalf of the caller, so the caller is responsible for calling `CFRelease` on the returned `CFString`.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

QTSoundDescriptionGetProperty

Gets a particular property of a sound description.

```
OSStatus QTSoundDescriptionGetProperty (
SoundDescriptionHandle     inDesc,
QTPropertyClass            inPropClass,
QTPropertyID               inPropID,
ByteCount                  inPropValueSize,
QTPropertyValuePtr         outPropValueAddress,
ByteCount                  *outPropValueSizeUsed);
```

Parameters

inDesc
The sound description being interrogated.

inPropClass
The class of the property being requested.

inPropID
The ID of the property being requested.

inPropValueSize
The size of the property value buffer.

outPropValueAddress
A pointer to the property value buffer.

outPropValueSizeUsed
The actual size of the returned property value (can be NULL).

Return Value

An error code. Returns noErr if there is no error.

Discussion

The following constants identify sound description properties.

```
enum {
    kQTSoundDescriptionPropertyID_AudioChannelLayout = 'clay',
    kQTSoundDescriptionPropertyID_MagicCookie = 'kuki',
    kQTSoundDescriptionPropertyID_AudioStreamBasicDescription = 'asbd',
    kQTSoundDescriptionPropertyID_UserReadableText = 'text'
};
```

Special Considerations

kQTSoundDescriptionPropertyID_AudioChannelLayout = 'clay'
Used to get or set an AudioChannelLayout value. This is a variable-size property because it may contain an array of Channel Descriptions. You must get the size by calling QTSoundDescriptionGetPropertyInfo, allocate a structure of that size, and then get the property.

kQTSoundDescriptionPropertyID_MagicCookie = 'kuki'
Used to get or set opaque bytes. This is a variable-size property, because it is completely defined by the codec that uses the cookie. You must get the size by calling QTSoundDescriptionGetPropertyInfo, allocate a structure of that size, and then get the property.

kQTSoundDescriptionPropertyID_AudioStreamBasicDescription = 'asbd'
Used to get an AudioStreamBasicDescription value.

`kQTSoundDescriptionPropertyID_UserReadableText = 'text'`
: Used to get a `CFStringRef` value. `QTSoundDescriptionGetProperty` does a `CFRetain` of the returned `CFString` on behalf of the caller, so the caller is responsible for calling `CFRelease` on the returned `CFString`.

`kQTAudioPropertyID_FormatString = 'fstr'`
: Used with `kQTPropertyClass_Audio` to get a `CFStringRef` value containing a localized, human readable string that describes an audio format; for example, "MPEG Layer 3." You may get this property from a `SoundDescription` handle by calling `QTSoundDescriptionGetProperty` or from a `StandardAudioCompression` (`scdi` or `audi`) component instance by calling `QTGetComponentProperty`.

`kQTAudioPropertyID_ChannelLayoutString = 'lstr'`
: Used with `kQTPropertyClass_Audio` to get a `CFStringRef` value containing a localized, human readable string that describes an audio channel layout; for example, "5.0 (L R C Ls Rs)." You may get this property from a `SoundDescription` handle by calling `QTSoundDescriptionGetProperty` or from a `StandardAudioCompression` (`scdi` or `audi`) component instance by calling `QTGetComponentProperty`.

`kQTAudioPropertyID_SampleRateString = 'rstr'`
: Used to get a `CFStringRef` value containing a localized, human readable string that describes an audio sample rate; for example, "44.100 kHz." You may get this property from a `SoundDescription` handle by calling `QTSoundDescriptionGetProperty` or from a `StandardAudioCompression` (`scdi` or `audi`) component instance by calling `QTGetComponentProperty`.

`kQTAudioPropertyID_SampleSizeString = 'sstr'`
: Used to get a `CFStringRef` value containing a localized, human readable string that describes an audio sample size; for example, "24-bit." This property will return a valid string only if the audio format is uncompressed (LPCM). You may get this property from a `SoundDescription` handle by calling `QTSoundDescriptionGetProperty` or from a `StandardAudioCompression` (`scdi` or `audi`) component instance by calling `QTGetComponentProperty`.

`kQTAudioPropertyID_BitRateString = 'bstr'`
: Used to get a `CFStringRef` value containing a localized, human readable string that describes an audio bit rate; for example, "12 kbps." You may get this property from a `StandardAudioCompression` (`scdi` or `audi`) component instance by calling `QTGetComponentProperty`.

`kQTAudioPropertyID_SummaryString = 'asum'`
: Used to get a `CFStringRef` value containing a localized, human readable string that summarizes an audio format; for example, "16-bit Integer (Big Endian), Stereo (L R), 48.000 kHz." You may get this property from a `SoundDescription` handle by calling `QTSoundDescriptionGetProperty` or from a `StandardAudioCompression` (`scdi` or `audi`) component instance by calling `QTGetComponentProperty`.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

QTSoundDescriptionSetProperty

Sets a particular property of a sound description.

```
OSStatus QTSoundDescriptionSetProperty (
SoundDescriptionHandle     inDesc,
QTPropertyClass            inPropClass,
QTPropertyID               inPropID,
ByteCount                  inPropValueSize,
ConstQTPropertyValuePtr   inPropValueAddress );
```

Parameters

inDesc
The sound description being modified.

inPropClass
The class of the property being set.

inPropID
The ID of the property being set.

inPropValueSize
The size of the property value buffer.

inPropValueAddress
A pointer to the property value buffer.

Return Value
An error code. Returns `noErr` if there is no error.

Discussion
The following constants identify sound description properties.

```
enum {
    kQTSoundDescriptionPropertyID_AudioChannelLayout = 'clay',
    kQTSoundDescriptionPropertyID_MagicCookie = 'kuki',
    kQTSoundDescriptionPropertyID_AudioStreamBasicDescription = 'asbd',
    kQTSoundDescriptionPropertyID_UserReadableText = 'text'
};
```

Special Considerations

`kQTSoundDescriptionPropertyID_AudioChannelLayout = 'clay'`
Used to get or set an `AudioChannelLayout` value. This is a variable-size property because it may contain an array of Channel Descriptions. You must get the size by calling `QTSoundDescriptionGetPropertyInfo`, allocate a structure of that size, and then get the property.

`kQTSoundDescriptionPropertyID_MagicCookie = 'kuki'`
Used to get or set opaque bytes. This is a variable-size property, because it is completely defined by the codec that uses the cookie. You must get the size by calling `QTSoundDescriptionGetPropertyInfo`, allocate a structure of that size, and then get the property.

`kQTSoundDescriptionPropertyID_AudioStreamBasicDescription = 'asbd'`
Used to get an `AudioStreamBasicDescription` value.

`kQTSoundDescriptionPropertyID_UserReadableText = 'text'`
Used to get a `CFStringRef` value. `QTSoundDescriptionGetProperty` does a `CFRetain` of the returned `CFString` on behalf of the caller, so the caller is responsible for calling `CFRelease` on the returned `CFString`.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

QTVisualContextCopyImageForTime

Retrieves an image buffer from the visual context, indexed by the provided time.

```
OSStatus QTVisualContextCopyImageForTime (
 QTVisualContextRef visualContext,
 CFAllocatorRef allocator,
 const CVTimeStamp *timeStamp,
 CVImageBufferRef *newImage );
```

Parameters

visualContext
: The visual context.

allocator
: Allocator used to create new CVImageBufferRef.

*CVTimeStamp *timeStamp*
: Time in question. Pass `NULL` to request the image at the current time.

newImage
: Points to variable to receive the new image.

Return Value
An error code. Returns `noErr` if there is no error.

Discussion
You should not request image buffers further ahead of the current time than the read-ahead time specified with the `kQTVisualContextExpectedReadAheadKey` attribute.You may skip images by passing later times, but you may not pass an earlier time than passed to a previous call to this function.

Version Notes
Introduced in QuickTime 7

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

QTVisualContextGetAttribute

Returns a visual context attribute.

```
OSStatus QTVisualContextGetAttribute (
 QTVisualContextRef visualContext,
 CFStringRef attributeKey,
 CFTypeRef *attributeValueOut );
```

Parameters

visualContext

The visual context.

attributeKey

Identifier of attribute to get.

attributeValueOut

A pointer to a variable that will receive the attribute value or `NULL` if the attribute is not set.

Return Value

An error code. Returns `noErr` if there is no error.

Discussion

This routine returns a visual context attribute.

Version Notes

Introduced in QuickTime 7

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

QTVisualContextGetTypeID

Returns the CFTypeID for QTVisualContextRef.

```
CFTypeID QTVisualContextGetTypeID (
 void );
```

Return Value

Undocumented.

Discussion

Use this function to test whether a CFTypeRef that extracted from a CF container such as a CFArray was a `QTVisualContextRef`.

Version Notes

Introduced in QuickTime 7

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

QTVisualContextIsNewImageAvailable

Queries whether a new image is available for a given time.

```
Boolean QTVisualContextIsNewImageAvailable (
 QTVisualContextRef visualContext,
 const CVTimeStamp *timeStamp );
```

Parameters

visualContext

The visual context.

*CVTimeStamp *timeStamp*

Time in question.

Return Value

A Boolean.

Discussion

This function returns `TRUE` if there is a image available for the specified time that is different from the last image retrieved from `QTVisualContextCopyImageForTime`.

Version Notes

Introduced in QuickTime 7

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

QTVisualContextSetAttribute

Sets a visual context attribute.

```
OSStatus QTVisualContextSetAttribute (
 QTVisualContextRef visualContext,
 CFStringRef attributeKey,
 CFTypeRef attributeValue );
```

Parameters

visualContext

The visual context.

attributeKey

Identifier of attribute to set

attributeValue

The value of the attribute to set, or `NULL` to remove a value.

Return Value

An error code. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 7

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

QTVisualContextSetImageAvailableCallback

Installs a user-defined callback to receive notifications when a new image becomes available.

```
OSStatus QTVisualContextSetImageAvailableCallback (
 QTVisualContextRef visualContext,
 QTVisualContextImageAvailableCallback imageAvailableCallback,
 void *refCon );
```

Parameters

visualContext

The visual context invoking the callback.

imageAvailableCallback

Time for which a new image has become available. May be `NULL`.

refCon

A user-defined value passed to `QTImageAvailableCallback`.

Return Value

An error code. Returns `noErr` if there is no error.

Discussion

Due to unpredictible activity, such as user seeks or the arrival of streaming video packets from a network, new images may become available for times supposedly occupied by previous images. Applications using the CoreVideo display link to drive rendering probably do not need to install a callback of this type, since they will already be checking for new images at a sufficient rate.

Version Notes

Introduced in QuickTime 7

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In

`ImageCompression.h`

QTVisualContextRelease

Releases a visual context object.

```
void QTVisualContextRelease ( QTVisualContextRef visualContext );
```

Parameters

visualContext

A reference to a visual context object. If you pass `NULL`, nothing happens.

Discussion

When the retain count decreases to zero the visual context is disposed.

Version Notes

Introduced in QuickTime 7

Availability

Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

QTVisualContextRetain

Retains a visual context object.

```
QTVisualContextRef QTVisualContextRetain (
 QTVisualContextRef visualContext );
```

Parameters

`visualContext`

A reference to a visual context object. If you pass `NULL`, nothing happens.

Return Value
On return, a reference to the same visual context object, for convenience.

Version Notes
Introduced in QuickTime 7

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

QTVisualContextTask

Causes visual context to release internally held resources for later re-use.

```
void QTVisualContextTask (
 QTVisualContextRef visualContext );
```

Parameters

`visualContext`

The visual context.

Discussion
For optimal resource management, this function should be called in every rendering pass, after old images have been released, new images have been used and all rendering has been flushed to the screen. This call is not mandatory.

Version Notes
Introduced in QuickTime 7

Availability
Carbon status: Supported C interface file: `ImageCompression.h`

Declared In
`ImageCompression.h`

SampleNumToMediaDecodeTime

Finds the decode time for a specified sample.

```
void SampleNumToMediaDecodeTime (
 Media          theMedia,
 SInt64         logicalSampleNum,
 TimeValue64    *sampleDecodeTime,
 TimeValue64    *sampleDecodeDuration );
```

Parameters

theMedia

The media for this operation. You obtain this media identifier from such functions as `NewTrackMedia` and `GetTrackMedia`.

logicalSampleNum

A 64-bit signed integer that contains the sample number.

sampleDecodeTime

A pointer to a time value. The function updates this time value to indicate the decode time of the sample specified by the *logicalSampleNum* parameter. This time value is expressed in the media's time scale. Set this parameter to `NULL` if you do not want this information.

sampleDecodeDuration

A pointer to a time value. The function updates this time value to indicate the decode duration of the sample specified by the *logicalSampleNum* parameter. This time value is expressed in the media's time scale. Set this parameter to `NULL` if you do not want this information.

Discussion

You can access this function's error returns through `GetMoviesError` and `GetMoviesStickyError`. It returns `paramErr` if there is a bad parameter value, or `noErr` if there is no error.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `Movies.h`

Declared In

`Movies.h`

SampleNumToMediaDisplayTime

Finds the display time for a specified sample.

```
void SampleNumToMediaDisplayTime (
 Media          theMedia,
 SInt64         logicalSampleNum,
 TimeValue64    *sampleDisplayTime,
 TimeValue64    *sampleDisplayDuration );
```

Parameters

theMedia

The media for this operation. You obtain this media identifier from such functions as `NewTrackMedia` and `GetTrackMedia`.

logicalSampleNum

A 64-bit signed integer that contains the sample number.

sampleDisplayTime

A pointer to a time value. The function updates this time value to indicate the display time of the sample specified by the *logicalSampleNum* parameter. This time value is expressed in the media's time scale. Set this parameter to `NULL` if you do not want this information.

sampleDisplayDuration

A pointer to a time value. The function updates this time value to indicate the display duration of the sample specified by the *logicalSampleNum* parameter. This time value is expressed in the media's time scale. Set this parameter to `NULL` if you do not want this information.

Discussion

You can access this function's error returns through `GetMoviesError` and `GetMoviesStickyError`. It returns `paramErr` if there is a bad parameter value, or `noErr` if there is no error.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `Movies.h`

Declared In

`Movies.h`

SCAudioInvokeLegacyCodecOptionsDialog

Invokes the legacy code options dialog of an audio codec component.

```
ComponentResult SCAudioInvokeLegacyCodecOptionsDialog (
 ComponentInstance    ci );
```

Parameters

ci

A component instance that identifies a connection to an audio codec component.

Return Value

An error code, or `noErr` if there is no error.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `QuickTimeComponents.h`

Declared In

`QuickTimeComponents.h`

SCCopyCompressionSessionOptions

Creates a compression session options object based upon the settings in the Standard Compression component.

```
ComponentResult SCCopyCompressionSessionOptions (
 ComponentInstance ci,
 ICMCompressionSessionOptionsRef *outOptions );
```

Parameters

ci

A component instance of Standard Compression component.

outOptions

On return, a reference to a new compression session options object.

Return Value

An error code. Returns `noErr` if there is no error. *paramErr* if the client did not set the `scAllowEncodingWithCompressionSession` preference flag.

Discussion

This function creates a new compression session options object using the compression settings of the Standard Compression component instance. You can use other Standard Compression component calls to set up the compression settings. Then you call this function to extract the compression settings in the form of a compression session options object. The returned object can be used to create a compression session object through `ICMCompressionSessionCreate()`.

The caller must indicate that he or she intends to use the new ICM compression session API to perform the compression operation, by setting the `scAllowEncodingWithCompressionSession` preference flag through `SCSetInfo()` with the `scPreferenceFlagsType` selector.

The caller of this function is expected to release the returned compression session options object through `ICMCompressionSessionOptionsRelease` when it is done.

Version Notes

Introduced in QuickTime 7

Availability

Carbon status: Supported C interface file: `QuickTimeComponents.h`

Declared In

`QuickTimeComponents.h`

SetDSequenceNonScheduledDisplayDirection

Sets the display direction for a decompress sequence.

```
OSErr SetDSequenceNonScheduledDisplayDirection (
 ImageSequence    sequence,
 Fixed            rate );
```

Parameters

sequence

Contains the unique sequence identifier that was returned by the `DecompressSequenceBegin` function.

rate

The display direction to be set. Negative values represent backward display and positive values represent forward display.

Return Value
An error code. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `QuickTimeComponents.h`

Declared In
`ImageCompression.h`

SetDSequenceNonScheduledDisplayTime

Sets the display time for a decompression sequence.

```
OSErr SetDSequenceNonScheduledDisplayTime (
 ImageSequence    sequence,
 TimeValue64      displayTime,
 TimeScale        displayTimeScale,
 UInt32           flags );
```

Parameters

`sequence`
Contains the unique sequence identifier that was returned by the `DecompressSequenceBegin` function.

`displayTime`
The display time to be set.

`displayTimeScale`
The display time scale to be set.

`flags`
Not used; set to 0.

Return Value
An error code. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `QuickTimeComponents.h`

Declared In
`ImageCompression.h`

SetMovieAudioBalance

Sets the balance level for the mixed audio output of a movie.

```
OSStatus SetMovieAudioBalance (
 Movie      m,
 Float32    leftRight,
 UInt32     flags );
```

Parameters

m

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromProperties`, `NewMovieFromFile`, and `NewMovieFromHandle`.

leftRight

A pointer to the new balance setting for the movie. The balance setting is a 32-bit floating-point value that controls the relative volume of the left and right sound channels. A value of 0 sets the balance to neutral. Positive values up to 1.0 shift the balance to the right channel, negative values up to –1.0 to the left channel.

flags

Not used; set to 0.

Return Value

An error code. Returns `noErr` if there is no error.

Discussion

The movie's balance setting is not stored in the movie; it is used only until the movie is closed. See `GetMovieAudioBalance` (page 80).

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `Movies.h`

Declared In

`Movies.h`

SetMovieAudioContext

Targets a movie to render into an audio context.

```
OSStatus SetMovieAudioContext (
 Movie      movie,
 QTAudioContextRef    audioContext;
```

Parameters

movie

The movie.

audioContext

The audio context that the movie will render into.

Return Value

An error code. Returns `noErr` if there is no error. .

Version Notes

Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

SetMovieAudioFrequencyMeteringNumBands

Configures frequency metering for a particular audio mix in a movie.

```
OSStatus SetMovieAudioFrequencyMeteringNumBands (
 Movie           m,
 FourCharCode    whatMixToMeter,
 UInt32          *ioNumBands );
```

Parameters

m

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromProperties`, `NewMovieFromFile`, and `NewMovieFromHandle`.

whatMixToMeter

The applicable mix of audio channels in the movie; see "`Movie Audio Mixes`" (page 268).

ioNumBands

A pointer to memory that stores the number of bands being metered. On calling this function, you specify the number of frequency bands you want to meter. If that number is higher than is possible (determined by factors such as the sample rate of the audio being metered), the function will return the number of bands it is actually going to meter. You can pass `NIL` or a pointer to 0 to disable metering.

Return Value
An error code. Returns `noErr` if there is no error.

Discussion
See `GetMovieAudioFrequencyMeteringNumBands` (page 82).

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

SetMovieAudioGain

Sets the audio gain level for the mixed audio output of a movie, altering the perceived volume of the movie's playback.

```
OSStatus SetMovieAudioGain (
 Movie      m,
 Float32    gain,
 UInt32     flags );
```

Parameters

m

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromProperties`, `NewMovieFromFile`, and `NewMovieFromHandle`.

gain

A 32-bit floating-point gain value of 0 or greater. 0.0 is silent, 0.5 is –6 dB, 1.0 is 0 dB (the audio from the movie is not modified), 2.0 is +6 dB, etc. The gain level can be set higher than 1.0 to allow quiet movies to be boosted in volume. Gain settings higher than 1.0 may result in audio clipping.

flags

Not used; set to 0.

Return Value
An error code. Returns `noErr` if there is no error.

Discussion
The movie gain setting is not stored in the movie; it is used only until the movie is closed. See `GetMovieAudioGain` (page 83).

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

SetMovieAudioMute

Sets the mute value for the audio mix of a movie currently playing.

```
OSStatus SetMovieAudioMute (
 Movie      m,
 Boolean    muted,
 UInt32     flags );
```

Parameters

m

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromProperties`, `NewMovieFromFile`, and `NewMovieFromHandle`.

muted

Pass `TRUE` to mute the movie audio, `FALSE` otherwise.

flags

Not used; set to 0.

Return Value
An error code. Returns `noErr` if there is no error.

Discussion
The movie mute setting is not stored in the movie; it is used only until the movie is closed. See `GetMovieAudioMute` (page 84).

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

SetMovieAudioVolumeMeteringEnabled

Enables or disables volume metering of a particular audio mix of a movie.

```
OSStatus SetMovieAudioVolumeMeteringEnabled (
 Movie          m,
 FourCharCode   whatMixToMeter,
 Boolean        enabled );
```

Parameters

m
The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromProperties`, `NewMovieFromFile`, and `NewMovieFromHandle`.

whatMixToMeter
The applicable mix of audio channels in the movie; see "`Movie Audio Mixes`" (page 268).

enabled
Pass `TRUE` to enable audio volume metering; pass `FALSE` to disable it.

Return Value
An error code. Returns `noErr` if there is no error.

Discussion
See `GetMovieAudioVolumeMeteringEnabled` (page 85).

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

SetMovieVisualBrightness

Sets the brightness adjustment for the movie.

```
OSStatus SetMovieVisualBrightness (
 Movie movie,
 Float32 brightness,
 UInt32 flags );
```

Parameters

movie

The movie.

brightness

New brightness adjustment.

flags

Reserved. Pass 0.

Return Value

An error code. Returns `noErr` if there is no error.

Discussion

The brightness adjustment for the movie. The value is a Float32 for which -1.0 means full black, 0.0 means no adjustment, and 1.0 means full white. The setting is not stored in the movie. It is only used until the movie is closed, at which time it is not saved.

Version Notes

Introduced in QuickTime 7

Availability

Carbon status: Supported C interface file: `Movies.h`

Declared In

`Movies.h`

SetMovieVisualContext

Targets a movie to render into a visual context.

```
OSStatus SetMovieVisualContext (
 Movie      movie,
 QTVisualContextRef    visualContext;
```

Parameters

movie

The movie.

visualContext

The visual context that the movie will render into. May be `NULL`..

Return Value

An error code. Returns `noErr` if there is no error. Returns `memFullErr` if memory cannot be allocated. Returns `kQTVisualContextNotAllowed` if the movie is not able to render using a visual context. Returns `paramErr` if the movie is `NULL`.

Discussion

When `SetMovieVisualContext` succeeds, it will retain the QTVisualContext object for its own use. If *visualContext* is `NULL`, the movie will not render any visual media. `SetMovieVisualContext` will fail if a different movie is already using the visual context, so you should first disassociate the other movie by calling `SetMovieVisualContext` with a `NULL` *visualContext*.

Special Considerations

Note that calling `SetMovieGWorld` on a movie that is connected to a visual context will work, but it may still keep a reference to the visual context. If you wish to completely disconnect the visual context, make sure to first call `SetMovieVisualContext` with a `NULL` *`visualContext`*.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

SetMovieVisualContrast

Sets the contrast adjustment for the movie.

```
OSStatus SetMovieVisualContrast (
 Movie movie,
 Float32 contrast,
 UInt32 flags );
```

Parameters

`movie`
The movie.

`contrast`
The new contrast adjustment.

`flags`
Reserved. Pass 0.

Return Value
An error code. Returns `noErr` if there is no error.

Discussion
The contrast adjustment for the movie. The value is a Float32 percentage (1.0f = 100%), such that 0.0 gives solid grey. The setting is not stored in the movie. It is only used until the movie is closed, at which time it is not saved.

Version Notes
Introduced in QuickTime 7

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

SetMovieVisualHue

Sets the hue adjustment for the movie.

```
OSStatus SetMovieVisualHue (
 Movie movie,
 Float32 hue,
 UInt32 flags );
```

Parameters

movie

The movie.

hue

New hue adjustment.

flags

Reserved. Pass 0.

Return Value

An error code. Returns `noErr` if there is no error.

Discussion

The hue adjustment for the movie. The value is a Float32 between -1.0 and 1.0, with 0.0 meaning no adjustment. This adjustment wraps around, such that -1.0 and 1.0 yield the same result. The setting is not stored in the movie. It is only used until the movie is closed, at which time it is not saved.

Version Notes

Introduced in QuickTime 7

Availability

Carbon status: Supported C interface file: `Movies.h`

Declared In

`Movies.h`

SetMovieVisualSaturation

Sets the color saturation adjustment for the movie.

```
OSStatus SetMovieVisualSaturation (
 Movie movie,
 Float32 saturation,
 UInt32 flags );
```

Parameters

movie

The movie.

saturation

The new saturation adjustment.

flags

Reserved. Pass 0.

Return Value

An error code. Returns `noErr` if there is no error.

Discussion

The color saturation adjustment for the movie. The value is a Float32 percentage (1.0f = 100%), such that 0.0 gives grayscale. The setting is not stored in the movie. It is only used until the movie is closed, at which time it is not saved.

Version Notes
Introduced in QuickTime 7

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

SetTrackAudioGain

Sets the audio gain level for the audio output of a track, altering the perceived volume of the track's playback.

```
OSStatus SetTrackAudioGain (
 Track      t,
 Float32    gain,
 UInt32     flags );
```

Parameters

`t`
: A track identifier, which your application obtains from such functions as `NewMovieTrack` and `GetMovieTrack`.

`gain`
: A 32-bit floating-point gain value of 0 or greater. 0.0 is silent, 0.5 is –6 dB, 1.0 is 0 dB (the audio from the track is not modified), 2.0 is +6 dB, etc. The gain level can be set higher than 1.0 to allow quiet tracks to be boosted in volume. Gain settings higher than 1.0 may result in audio clipping.

`flags`
: Not used; set to 0.

Return Value
An error code. Returns `noErr` if there is no error.

Discussion
The track's gain setting is not stored in the movie; it is used only until the movie is closed. See `GetTrackAudioGain` (page 89).

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

SetTrackAudioMute

Mutes or unmutes the audio output of a track.

```
OSStatus SetTrackAudioMute (
 Track      t,
 Boolean    muted,
 UInt32     flags );
```

Parameters

t

A track identifier, which your application obtains from such functions as `NewMovieTrack` and `GetMovieTrack`.

muted

Pass `TRUE` to mute the track's audio, `FALSE` to unmute it.

flags

Not used; set to 0.

Return Value

An error code. Returns `noErr` if there is no error.

Discussion

The track mute setting is not stored in the movie; it is used only until the movie is closed. See `GetTrackAudioMute` (page 89).

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported C interface file: `Movies.h`

Declared In

`Movies.h`

TrackTimeToMediaDisplayTime

Converts a track's time value to a display time value that is appropriate to the track's media, using the track's edit list.

```
TimeValue64 TrackTimeToMediaDisplayTime (
 TimeValue64    value,
 Track          theTrack );
```

Parameters

value

A 64-bit time value that represents the track's time value; it must be expressed in the time scale of the movie that contains the track.

theTrack

A track identifier, which your application obtains from such functions as `NewMovieTrack` and `GetMovieTrack`.

Return Value

A 64-bit time value that represents the corresponding time in media display time, in the media's time coordinate system. If the track time corresponds to empty space, this function returns a value of –1.

Discussion
This function maps the track time through the track's edit list to come up with the media time. This time value contains the track's time value according to the media's time coordinate system. If the time you specified lies outside of the movie's active segment or corresponds to empty space in the track, this function returns a value of –1. Hence you can use it to determine whether a specified track edit is empty.

Version Notes
Introduced in QuickTime 7. This function is a 64-bit replacement for `TrackTimeToMediaTime`.

Availability
Carbon status: Supported C interface file: `Movies.h`

Declared In
`Movies.h`

Callbacks

The callback functions new to the QuickTime 7 API are documented alphabetically in this section.

ICMDecompressionTrackingCallbackProc

The callback through which a client of an ICM decompression session receives decoded frames and information about decoding.

```
typedef void (*ICMDecompressionTrackingCallback)(
void *decompressionTrackingRefCon, OSStatus result,  ICMDecompressionTrackingFlags
 decompressionTrackingFlags, CVPixelBufferRef  pixelBuffer, TimeValue64 displayTime,
TimeValue64 displayDuration, ICMValidTimeFlags validTimeFlags,
void *sourceFrameRefCon, void *reserved);

// Declaration of a typical application-defined function
Boolean MyICMDecompressionTrackingCallbackProc (
    void                             *decompressionTrackingRefCon,
    OSStatus                         result,
    ICMDecompressionTrackingFlags    decompressionTrackingFlags,
    CVPixelBufferRef                 pixelBuffer,
    TimeValue64                      displayTime,
    TimeValue64                      displayDuration,
    ICMValidTimeFlags                validTimeFlags,
    void                             *sourceFrameRefCon,
    void                             *reserved );
```

Parameters

decompressionTrackingRefCon
The callback's reference value, copied from the *decompressionTrackingRefCon* field of an `ICMDecompressionTrackingCallbackRecord` (page 258) structure.

result
Indicates whether there was an error in decompression.

decompressionTrackingFlags

One or more flags describing the a frame's state transitions:

`kICMDecompressionTracking_LastCall = 1L<<0`

This is the last call for this *sourceFrameRefCon*.

`kICMDecompressionTracking_ReleaseSourceData = 1L<<1`

The session no longer needs the source data pointer.

`kICMDecompressionTracking_EmittingFrame = 1L<<2`

A frame is being emitted. The *pixelBuffer* parameter contains the decompressed frame. If the decompression session is targetting a visual context, the frame has not yet been sent to the visual context but will be sent after the callback returns.

`kICMDecompressionTracking_FrameDecoded = 1L<<3`

This frame was decoded.

`kICMDecompressionTracking_FrameDropped = 1L<<4`

The codec decided to drop this frame.

`kICMDecompressionTracking_FrameNeedsRequeueing = 1L<<5`

This frame will not be able to be displayed unless it is queued for redecode (this constant is also known as `FrameNotDisplayable`).

pixelBuffer

When the `kICMDecompressionTracking_EmittingFrame` flag is set in *decompressionTrackingFlags*, this parameter must reference a pixel buffer containing the decompressed frame.

displayTime

If `kICMValidTime_DisplayTimeStampIsValid` is set in *validTimeFlags*, this parameter must pass the display time of the frame.

displayDuration

If `kICMValidTime_DisplayDurationIsValid` is set in *validTimeFlags*, this parameter must pass the display duration of the frame.

validTimeFlags

Indicates which of *displayTime* and *displayDuration* is valid:

`kICMValidTime_DisplayTimeStampIsValid`

The time value passed in *displayTimeStamp* is valid.

`kICMValidTime_DisplayDurationIsValid`

The time value passed in *displayDuration* is valid.

sourceFrameRefCon

The frame's reference value, copied from the *sourceFrameRefCon* parameter passed to `ICMDecompressionSessionDecodeFrame` (page 142).

reserved

Reserved for future use.

Discussion

This callback is referenced by an `ICMDecompressionTrackingCallbackRecord` (page 258).

MovieExportStageReachedCallbackProc

Installed by NewMovieExportStageReachedCallbackUPP (page 190).

```
typedef OSErr (*MovieExportStageReachedCallbackProcPtr)(OSType inStage, Movie
inMovie, ComponentInstance inDataHandler, Handle inDataRef,
OSType inDataRefType, void *refCon);

// Declaration of a typical application-defined function
Boolean MyMovieExportStageReachedCallbackProc (
    OSType                 inStage,
    Movie                  inMovie,
    ComponentInstance      inDataHandler,
    Handle                 inDataRef,
    OSType                 inDataRefType,
    void                   *refCon );
```

Parameters

inStage

A movie export stage.

inMovie

A movie.

inDataHandler

A data handler component.

inDataRef

A handle to a data reference.

inDataRefType

The type of the data reference.

refCon

A reference constant to be passed to the callback specified in NewMovieExportStageReachedCallbackUPP (page 190). Use this parameter to point to a data structure containing any information your callback needs.

SGAudioCallbackProc

Provides access to a SGAudioMediaType channel's data at various point along the signal flow.

```
typedef OSStatus (*SGAudioCallbackProcPtr)
(SGChannel c, void  *inRefCon, SGAudioCallbackFlags *ioFlags, const  AudioTimeStamp
 *inTimeStamp, const UInt32 *inNumberPackets,
const AudioBufferList *inData, const AudioStreamPacketDescription
*inPacketDescriptions);
```

```
// Declaration of a typical application-defined function
OSStatus MySGAudioCallbackProc (
    SGChannel                           c,
    void                                *inRefCon,
    SGAudioCallbackFlags                *ioFlags,
    const AudioTimeStamp                *inTimeStamp,
    const UInt32                        *inNumberPackets,
    const AudioBufferList               *inData,
    const AudioStreamPacketDescription  *inPacketDescriptions );
```

Parameters

c

The sequence grabber channel that has originating this callback.

inRefCon

A reference constant passed by the caller. Use this parameter to point to a data structure containing any information your callback needs.

ioFlags

Currently not used.

inTimeStamp

The time stamp associated with the first sample passed in *inData*.

inNumberPackets

The number of data packets held in *inData*. With LPCM formats the number of packets is the same as number of frames.

inData

A bufferlist containing the requested sample data.

inPacketDescriptions

If the packets contained in *inData* are of variable size, this parameter should pass an array of *inNumberPackets* packet descriptions.

Discussion

Use `QTSetComponentProperty` with `kQTPropertyClass_SGAudio` and any of the following property IDs to specify which callback you would like to receive:

- `kQTSGAudioPropertyID_PreMixCallback`
- `kQTSGAudioPropertyID_PostMixCallback`
- `kQTSGAudioPropertyID_PreConversionCallback`
- `kQTSGAudioPropertyID_PostConversionCallback`

Pass an `SGAudioCallbackStruct` (page 261)as the data payload. Clients define an `SGAudioCallbackProc` in order to tap into a `SGAudioMediaType` channel, gaining access to its data at various points along the signal flow chain. Clients should be aware that they may be called back on threads other than the thread on which they registered for the callback. They should do as little work as possible inside their callback, returning control as soon as possible to the channel.

QTOpenGLTextureAvailableCallbackProc

Receives notifications when a new OpenGL texture becomes available.

```
typedef void (*QTOpenGLTextureAvailableCallback)( QTOpenGLTextureContextRef
textureContext, const CVTimeStamp *timeStamp, void *refCon );

// Declaration of a typical application-defined function
OSStatus MyQTOpenGLTextureAvailableCallback (
    QTOpenGLTextureContextRef    textureContext,
    const CVTimeStamp            *timeStamp,
    void                         *refCon );
```

Parameters

textureContext

The OpenGL texture context invoking the callback.

timeStamp

Time for which a new texture has become available.

refCon

A reference constant passed by the caller. Use this parameter to point to a data structure containing any information your callback needs.

Discussion

Due to unpredictible activity, such as user seeks or the arrival of streaming video packets from a network, new textures may become available for times supposedly occupied by previous textures. Responsive applications, therefore, should use this callback to discover as soon as possible when a movie needs to be updated.

Data Structures

The public data structures new to the QuickTime 7 API are documented alphabetically in this section. Certain other data structures are referenced by QuickTime 7 functions but are opaque.

ICMDecompressionTrackingCallbackRecord

Designates a tracking callback for an ICM decompression session.

```
struct ICMDecompressionTrackingCallbackRecord {
ICMDecompressionTrackingCallback    decompressionTrackingCallback;
void                                *decompressionTrackingRefCon;
};
```

Fields

`decompressionTrackingCallback`

The callback function pointer. See `ICMDecompressionTrackingCallbackProc` (page 254).

`decompressionTrackingRefCon`

The callback's reference value.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported; C interface file: `ImageCompression.h`

ICMMultiPassStorageCallbacks

Designates a collection of callbacks for creating a custom multipass storage object.

```
struct ICMMultiPassStorageCallbacks {
UInt32                                   version;
void                                     *storageRefCon;
ICMMultiPassSetDataAtTimeStampCallback    setDataAtTimeStampCallback;
ICMMultiPassGetTimeStampCallback          getTimeStampCallback;
ICMMultiPassCopyDataAtTimeStampCallback   copyDataAtTimeStampCallback;
ICMMultiPassReleaseCallback               releaseCallback;
};
```

Fields

`version`
The version of this structure. Set to `kICMMultiPassStorageCallbacksVersionOne`.

`storageRefCon`
A pointer to a reference constant. Use this parameter to point to a data structure containing any information your callback needs.

`setDataAtTimeStampCallback`
A callback for storing values.

`getTimeStampCallback`
A callback for finding time stamps.

`copyDataAtTimeStampCallback`
A callback for retrieving values.

`releaseCallback`
A callback for disposing the callback's state when done.

Discussion
This structure is used by `ICMMultiPassStorageCreateWithCallbacks` (page 173).

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported; C interface file: `ImageCompression.h`

QTAudioFrequencyLevels

Stores the frequency meter level settings for the audio channels in a movie mix.

```
struct QTAudioFrequencyLevels {
 UInt32     numChannels;
 UInt32     numFrequencyBands;
 Float32    level[1];
};
```

Fields

`numChannels`
The number of audio channels.

`numFrequencyBands`
The number of frequency bands for each channel.

`level`

A 32-bit floating-point value for each frequency band. The frequency bands for each channel are stored contiguously, with all the band levels for the first channel first, all the band levels for the second channel next, etc. The total number of 32-bit values in this field equals `numFrequencyBands` times `numChannels`.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported; C interface file: `Movies.h`

Platform Considerations

Associated function: `GetMovieAudioFrequencyLevels` (page 81)

QTAudioVolumeLevels

Stores the volume level settings for the audio channels in a movie mix.

```
struct QTAudioVolumeLevels {
 UInt32     numChannels;
 Float32    level[1];
};
```

Fields

`numChannels`

The number of audio channels.

`level`

A 32-bit floating-point value for each channel's volume.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported; C interface file: `Movies.h`

Platform Considerations

Associated function: `GetMovieAudioVolumeLevels` (page 84)

QTNewMoviePropertyElement

Stores a movie property for `NewMovieFromProperties` (page 191).

```
struct QTNewMoviePropertyElement {
QTPropertyClass      propClass;
QTPropertyID         propID;
ByteCount            propValueSize;
QTPropertyValuePtr   propValueAddress;
OSStatus             propStatus;
};
```

Fields

`propClass`
: A four-character code designating the class of a movie property. See "New Movie Property Codes" (page 285).

`propID`
: The ID of the property.

`propValueSize`
: The size in bytes of the property passed in `propValueAddress`.

`propValueAddress`
: A pointer to a movie property. Since the data type is fixed for each element's property class and ID, these is no ambiguity about the data type for its property value.

`propStatus`
: Indicates any problems with the property. For example, if a property is not understood by the function it is passed to, this field is set appropriately. See the discussion in `NewMovieFromProperties` (page 191).

Discussion

When you call `NewMovieFromProperties`, you allocate and own arrays of these elements to pass to it, as well as the property values that each element points to. You are responsible for disposing of all of these memory allocations.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported; C interface file: `Movies.h`

Platform Considerations

Associated function: `NewMovieFromProperties` (page 191)

SGAudioCallbackStruct

Used to call an `SGAudioCallbackProc` (page 256).

```
struct SGAudioCallbackStruct {
  SGAudioCallback     inputProc;
  void                *inputProcRefCon;
};
```

Fields

`inputProc`
: An `SGAudioCallbackProc` (page 256).

`inputProcRefCon`

A reference constant. Use this parameter to point to a data structure containing any information your callback needs.

Version Notes

Introduced in QuickTime 7.

Availability

Carbon status: Supported; C interface file: `Movies.h`

SoundDescriptionV2

Provides version 2 of the `SoundDescription` data structure.

```
struct SoundDescriptionV2 {
 SInt32     descSize;
 OSType     dataFormat;
 SInt32     resvd1;
 SInt16     resvd2;
 SInt16     dataRefIndex;
 SInt16     version;
 SInt16     revlevel;
 SInt32     vendor;
 SInt16     always3;
 SInt16     always16;
 SInt16     alwaysMinus2;
 SInt16     always0;
 UInt32     always65536;
 UInt32     sizeOfStructOnly;
 Float64    audioSampleRate;
 UInt32     numAudioChannels;
 SInt32     always7F000000;
 UInt32     constBitsPerChannel;
 UInt32     formatSpecificFlags;
 UInt32     constBytesPerAudioPacket;
 UInt32     constLPCMFramesPerAudioPacket;
            /* additional atom-based extensions ([long size, long type,
            some data], repeat) */
};
```

Fields

`descSize`

Total size of this structure, including extra data.

`dataFormat`

Set to `'lpcm'` for uncompressed data; otherwise set to the compression type. For a list of compression type codes, see the QuickTime API Reference.

`resvd1`

Reserved; set to 0.

`resvd2`

Reserved; set to 0.

`dataRefIndex`

Reserved; set to 0.

`version`
: Version of this structure; set to 2.

`revlevel`
: Set to codec version number.

`always3`
: Reserved; set to 3.

`always16`
: Reserved; set to 16 (0x0010).

`alwaysMinus2`
: Reserved; set to –2 (0xFFFE).

`always0`
: Reserved; set to 0.

`always65536`
: Reserved; set to 65536 (0x00010000).

`sizeOfStructOnly`
: Set to `sizeof(SoundDescriptionV2)`, equivalent to the offset to any structure extensions.

`audioSampleRate`
: Set to a 64-bit floating-point number representing the number of audio frames per second; for example, 44100.0.

`numAudioChannels`
: Set to the number of audio channels; any channel assignment info will be in an extension.

`always7F000000`
: Reserved; set to 7F000000.

`constBitsPerChannel`
: Set to the number of bits per channel only if this value is constant and the audio is uncompressed. Otherwise set to 0.

`formatSpecificFlags`
: See LPCM flag definitions in `CoreAudioTypes.h`.

`constBytesPerAudioPacket`
: Set to the number of bytes per packet only if this value is constant. Otherwise set to 0.

`constLPCMFramesPerAudioPacket`
: Set to the number of PCM frames per packet only if this value is constant. Otherwise set to 0.

Version Notes
Introduced in QuickTime 7.

Availability
Carbon status: Supported; C interface file: `Movies.h`

Platform Considerations

You should never have to know this definition, except for debugging purposes. Use the new QuickTime sound description APIs to treat sound descriptions as if they are opaque.

Constants

This section lists constants that are newly defined in QuickTime 7.

ICM Compression Session Options

The following values are used to select options for ICM compression session objects:

```
kQTPropertyClass_ICMCompressionSessionOptions = 'icso',
kICMCompressionSessionOptionsPropertyID_AllowAsyncCompletion = 'asok',
kICMCompressionSessionOptionsPropertyID_AllowFrameReordering = 'b ok',
kICMCompressionSessionOptionsPropertyID_AllowFrameTimeChanges = '+ ok',
kICMCompressionSessionOptionsPropertyID_AllowTemporalCompression = 'p ok',
kICMCompressionSessionOptionsPropertyID_AverageDataRate = 'aver',
kICMCompressionSessionOptionsPropertyID_ColorTable = 'clut',
kICMCompressionSessionOptionsPropertyID_CompressorComponent = 'imco',
kICMCompressionSessionOptionsPropertyID_CompressorSettings = 'cost',
kICMCompressionSessionOptionsPropertyID_CPUTimeBudget = 'cput',
kICMCompressionSessionOptionsPropertyID_DataRateLimitCount = 'har#',
kICMCompressionSessionOptionsPropertyID_DataRateLimits = 'hard',
kICMCompressionSessionOptionsPropertyID_Depth = 'deep',
kICMCompressionSessionOptionsPropertyID_DurationsNeeded = 'need',
kICMCompressionSessionOptionsPropertyID_MaxDataRateLimits = 'mhar',
kICMCompressionSessionOptionsPropertyID_MaxFrameDelayCount = 'cwin',
kICMCompressionSessionOptionsPropertyID_MaxFrameDelayTime = 'cwit',
kICMCompressionSessionOptionsPropertyID_MaxKeyFrameInterval = 'kyfr',
kICMCompressionSessionOptionsPropertyID_MultiPassStorage = 'imps',
kICMCompressionSessionOptionsPropertyID_Quality = 'qual',
kICMCompressionSessionOptionsPropertyID_SourceFrameCount = 'frco',
kICMCompressionSessionOptionsPropertyID_WasCompressed = 'wasc'
```

`kQTPropertyClass_ICMCompressionSessionOptions = 'icso'`
: Class identifier for compression session option object properties.

`kICMCompressionSessionOptionsPropertyID_AllowAsyncCompletion = 'asok'`
: Enables the compressor to call the encoded-frame callback from a different thread. By default this option is `FALSE`, which means that the compressor must call the encoded-frame callback from the same thread as `ICMCompressionSessionEncodeFrame` and `ICMCompressionSessionCompleteFrames`.

`kICMCompressionSessionOptionsPropertyID_AllowFrameReordering = 'b ok'`
: Enables frame reordering. To encode B-frames a compressor must reorder frames, which may mean that the order in which they are emitted and stored (the decode order) may be different from the order in which they are presented to the compressor (the display order). By default, frame reordering is disabled. To encode using B-frames, you must enable frame reordering by passing `TRUE` in this property.

`kICMCompressionSessionOptionsPropertyID_AllowFrameTimeChanges = '+ ok'`
: Enables the compressor to modify frame times, improving its performance. Some compressors are able to identify and coalesce runs of identical frames and emit single frames with longer duration, or emit frames at a different frame rate from the original. By default, this flag is set to `FALSE`, which forces the compressor to emit one encoded frame for every source frame and to preserve frame display times. This option replaces the practice of having compressors return special high similarity values to indicate that frames can be dropped.

`kICMCompressionSessionOptionsPropertyID_AllowTemporalCompression = 'p ok'`
: Enables temporal compression of P-frames and B-frames. By default, temporal compression is disabled.

`kICMCompressionSessionOptionsPropertyID_AverageDataRate = 'aver'`
: The long-term desired average data rate in bytes per second. This is not an absolute limit. The default data rate is zero, indicating that the setting of `kICMCompressionSessionOptionsPropertyID_Quality` should determine the size of compressed

data. Data rate settings have effect only when timing information is provided for source frames. Some codecs do not accept limiting to specified data rates.

`kICMCompressionSessionOptionsPropertyID_ColorTable = 'clut'`
: The color table for compression, used with indexed-color depths. Clients who are passed this property are responsible for disposing the returned `CTabHandle`.

`kICMCompressionSessionOptionsPropertyID_CompressorComponent = 'imco'`
: Sets a specific compressor component or component instance to be used, or passes one of the wildcards `anyCodec`, `bestSpeedCodec`, `bestFidelityCodec`, or `bestCompressionCodec`. Pass this option to force the Image Compression Manager to use a specific compressor component or compressor component instance. To allow the Image Compression Manager to choose the compressor component, set the compressorComponent to `anyCodec` (the default), `bestSpeedCodec`, `bestFidelityCodec`, or `bestCompressionCodec`. If you pass in a component instance that you opened, the ICM will not close that instance; you must do so after the compression session is released.

`kICMCompressionSessionOptionsPropertyID_CompressorSettings = 'cost'`
: A handle containing compressor settings. The compressor will be configured with these settings (by a call to `ImageCodecSetSettings`) during the `ICMCompressionSessionCreate` process.

`kICMCompressionSessionOptionsPropertyID_CPUTimeBudget = 'cput'`
: Recommends a CPU time budget for a compressor in microseconds per frame. Zero means to go as fast as possible. By default, this is set to `kICMUnlimitedCPUTimeBudget`, which sets no limit. This option provides only an advisory hint, and some compressors may ignore it. Compressors are not compelled to use the full time budget if they complete ahead of time. Multithreaded compressors may use this amount of CPU time on each processor.

`kICMCompressionSessionOptionsPropertyID_DataRateLimitCount = 'har#'`
: The current number of data rate limits.

`kICMCompressionSessionOptionsPropertyID_DataRateLimits = 'hard'`
: Zero, one, or two hard limits on data rate. Each hard limit is described by a data size in bytes and a duration in seconds. It requires that the total size of compressed data for any contiguous segment of that duration (in decode time) must not exceed the data size. By default, no data rate limits are set. When setting this property, the *`inPropValueSize`* parameter should be the number of data rate limits multiplied by `sizeof(ICMDataRateLimit)`. Data rate settings have an effect only when timing information is provided for source frames. Some codecs do not accept limiting to specified data rates.

`kICMCompressionSessionOptionsPropertyID_Depth = 'deep'`
: The depth for compression. If a compressor does not support a specific depth, the closest supported depth will be used, preferring deeper depths to shallower depths. The default depth is `k24RGBPixelFormat`.

`kICMCompressionSessionOptionsPropertyID_DurationsNeeded = 'need'`
: Indicates that durations of emitted frames are needed. If this option is set and source frames are provided with times but not durations, then frames will be delayed so that durations can be calculated as the difference between one frame's time stamp and the next frame's time stamp. By default, this flag is `FALSE`, so frames will not be delayed in order to calculate durations. If you pass encoded frames to `AddMediaSampleFromEncodedFrame`, you must set this flag to `TRUE`.

`kICMCompressionSessionOptionsPropertyID_MaxDataRateLimits = 'mhar'`
: The maximum allowed number of data rate limits, currently 2.

`kICMCompressionSessionOptionsPropertyID_MaxFrameDelayCount = 'cwin'`
: The maximum frame delay count is the maximum number of frames that a compressor is allowed to hold before it must output a compressed frame. This value limits the number of frames that may be held in the "compression window." If the maximum frame delay count is M, then before the call to

encode frame N returns, frame N-M must have been emitted. The default value is `kICMUnlimitedFrameDelayCount`, which sets no limit on the compression window.

`kICMCompressionSessionOptionsPropertyID_MaxFrameDelayTime = 'cwit'`
: The maximum frame delay time is the maximum difference between a source frame's display time and the corresponding encoded frame»s decode time. This value limits the span of display time that may be held in the "compression window." If the maximum frame delay time is TM, then before the call to encode a frame with display time TN returns, all frames with display times up to and including TN-TM must have been emitted. The default value is `kICMUnlimitedFrameDelayTime`, which sets no time limit on the compression window.

`kICMCompressionSessionOptionsPropertyID_MaxKeyFrameInterval = 'kyfr'`
: The maximum interval between key frames, also known as the key frame rate. Compressors are allowed to generate key frames more frequently if this would result in more efficient compression. The default key frame interval is 0, which indicates that the compressor should choose where to place all key frames. This differs from previous practice, in which a key frame rate of zero disabled temporal compression.

`kICMCompressionSessionOptionsPropertyID_MultiPassStorage = 'imps'`
: A multipass compression client must provide a storage location for multipass data. Pass `ICMMultiPassStorageCreateWithTemporaryFile` to make the ICM store multipass data in a temporary file. Pass `ICMMultiPassStorageCreateWithCallbacks` to manage the storage yourself. Note that the amount of multipass data to be stored can be substantial; it could be greater than the size of the output movie file. If this property is not `NULL`, the client must call `ICMCompressionSessionBeginPass` and `ICMCompressionSessionEndPass` around groups of calls to `ICMCompressionSessionEncodeFrame`. By default, this property is `NULL` and multipass compression is not enabled. The compression session options object retains the multipass storage object when one is set.

`kICMCompressionSessionOptionsPropertyID_Quality = 'qual'`
: The compression quality. This value is always used to set the spatial quality; if temporal compression is enabled, it is also used to set temporal quality. The default quality is `codecNormalQuality`.

`kICMCompressionSessionOptionsPropertyID_SourceFrameCount = 'frco'`
: Indicates the number of source frames, if known. If nonzero, this value should equal the exact number of times that the client calls `ICMCompressionSessionEncodeFrame` in each pass. The default is 0, which indicates that the number of source frames is not known.

`kICMCompressionSessionOptionsPropertyID_WasCompressed = 'wasc'`
: Indicates that the source was previously compressed. This property is an optional information hint to the compressor; by default it is `FALSE`.

ICM Compression Session Properties

The following constants represent properties of ICM compression sessions:

```
kQTPropertyClass_ICMCompressionSession = 'icse',
kICMCompressionSessionPropertyID_CompressorPixelBufferAttributes = 'batt',
kICMCompressionSessionPropertyID_ImageDescription = 'idsc',
kICMCompressionSessionPropertyID_PixelBufferPool = 'pool',
kICMCompressionSessionPropertyID_TimeScale = 'tscl'
```

`kQTPropertyClass_ICMCompressionSession = 'icse'`
: Class identifier for compression session properties.

`kICMCompressionSessionPropertyID_CompressorPixelBufferAttributes = 'batt'`
: The compressor's pixel buffer attributes for the compression session. You can use these to create a pixel buffer pool for source pixel buffers. This is not the same as the `sourcePixelBufferAttributes` property passed to `ICMCompressionSessionCreate`. Getting this property does not change its retain count.

`kICMCompressionSessionPropertyID_ImageDescription = 'idsc'`
: The image description for a compression session. For some codecs, the image description may not be available before the first frame is compressed. Multiple calls to retrieve this property will return the same handle. The ICM will dispose of this handle when the compression session is disposed; the caller must not dispose of it.

`kICMCompressionSessionPropertyID_PixelBufferPool = 'pool'`
: A pool that can provide ideal source pixel buffers for a compression session. The compression session creates this pixel buffer pool based on the compressor's pixel buffer attributes and any pixel buffer attributes passed in to `ICMCompressionSessionCreate`. If the source pixel buffer attributes and the compressor pixel buffer attributes can not be reconciled, the pool is based on the source pixel buffer attributes and the ICM converts each `CVPixelBuffer` internally.

`kICMCompressionSessionPropertyID_TimeScale = 'tscl'`
: The time scale for the compression session.

Visual Context Types

The following are values for `kQTVisualContextTypeKey`, a read-only `CFStringRef` that defines the type of the visual context:

`kQTVisualContextType_PixelBuffer`
: The value of `kQTVisualContextTypeKey` for pixel buffer visual contexts.

`kQTVisualContextType_OpenGLTexture`
: The value of `kQTVisualContextTypeKey` for OpenGL texture visual contexts.

`kQTVisualContextColorSpaceKey`
: A `CGColorSpaceRef` that defines the color space of images produced by a visual context. If this attribute is not set, images may use any color space.

`kQTVisualContextExpectedReadAheadKey`
: A `CFNumberRef` that defines the number of seconds ahead of real time that the client expects to pull images out of a visual context. Applications using the Core Video display link should set this attribute according to the value returned by `CVDisplayLinkGetOutputVideoLatency`.

`kQTVisualContextPixelBufferAttributesKey`
: A `CFDictionaryRef` that defines the dictionary containing pixel buffer attributes. See `kICMCompressionSessionPropertyID_PixelBufferPool` in `ICM Compression Session Properties` (page 266).

`kQTVisualContextTargetDimensionsKey`
: A `CFDictionaryRef` that defines the dictionary containing `kQTVisualContextTargetDimensions_WidthKey` and `kQTVisualContextTargetDimensions_HeightKey` values (see below). This key is used as a hint to optimize certain media types, such as text, that can be rendered at any resolution. If this attribute is not set, the movie will be rendered at its native resolution.

`kQTVisualContextTargetDimensions_WidthKey`
: A `CFNumberRef` that defines the width, in pixels, of the rendering target.

`kQTVisualContextTargetDimensions_HeightKey`
: A `CFNumberRef` that defines the height, in pixels, of the rendering target.

Movie Audio Mixes

Three new four-character constants define the mix of audio channels for several functions:

```
kQTAudioMeter_StereoMix = 'stmx'
kQTAudioMeter_DeviceMix = kQTAudioPropertyID_DeviceChannelLayout = 'dcly'
kQTAudioMeter_MonoMix = 'momx'
```

`kQTAudioMeter_StereoMix`
: Meter a stereo (two-channel) mix of the enabled sound tracks in the movie. This option is offered only for MovieAudioFrequencyMetering.

`kQTAudioMeter_DeviceMix`
: Meter the movie's mix to the `AudioChannelLayout` of the device the movie is playing to. To determine the channel layout of this mix, you call the `kAudioPropertyID_DeviceChannelLayout` movie property.

`kQTAudioMeter_MonoMix`
: Meter the movie as if it had been mixed to monaural. This option is offered only for MovieAudioFrequencyMetering.

The constants listed above are passed by the following QuickTime 7 functions:

GetMovieAudioFrequencyLevels
GetMovieAudioFrequencyMeteringBandFrequencies
GetMovieAudioFrequencyMeteringNumBands
GetMovieAudioVolumeLevels
GetMovieAudioVolumeMeteringEnabled
SetMovieAudioFrequencyMeteringNumBands
SetMovieAudioVolumeMeteringEnabled

Audio Property Selectors

The following values are used as `ComponentPropertyID` selectors. Use these with the StandardCompressionSubTypeAudio (`'scdi'`/`'audi'`) component. All property IDs are to be used in conjunction with the `kQTPropertyClass_SCAudio` property class.

`kQTSCAudioPropertyID_AvailableCompressionFormatList = 'acf#'`
: A read/listen C-style array of `OSType` values that specifies the list of available output compression formats. This list includes all the `kAudioEncoderComponentType` components and `kSoundCompressor` type components on the user's system. You can restrict the list by using the `kQTSCAudioPropertyID_CompressionFormatList` property. Use `QTGetComponentPropertyInfo` to discover the number of bytes you should allocate for this array.

`kQTSCAudioPropertyID_ClientRestrictedCompressionFormatList = 'crf#'`
: A read/write/listen C-style array of `OSType` values that specifies a client-restricted set of output compression formats that you should list as available. Use `QTGetComponentPropertyInfo` to discover the number of bytes you should allocate to hold this array.

`kQTSCAudioPropertyID_AvailableCompressionFormatNamesList = 'cnm#'`
: A read/write `CFArrayRef` structure of `CFStringRef` structures that reference the human-readable names of each item in a `kQTSCAudioPropertyID_AvailableCompressionFormatList`. The caller assumes responsibility for calling `CFRelease` to dispose of the `CFArrayRef` structure.

`kQTSCAudioPropertyID_HasLegacyCodecOptionsDialog = 'opn?'`
: Some compression formats have format-specific properties that are accessible only via a compressor-provided dialog. This constant specifies a read/listen `Boolean` value that lets you know if the current compression format has such a dialog.

`kQTSCAudioPropertyID_ConstantBitRateFormatsOnly = '!vbr'`
: By default, constant as well as variable bit rate compression formats are shown in the available format list. This constant specifies a read/write/listen `Boolean` value that lets you restrict the available formats to constant bit rate formats by setting this property to `TRUE`.

`kQTSCAudioPropertyID_AvailableSampleRateList = 'avr#'`
: A read/listen C-style array of `AudioValueRange` values that specifies a list of available output sample rates. This list is specific to the compression format and takes into account any restrictions imposed by a client using the `kQTSCAudioPropertyID_ClientRestrictedSampleRateList` property. Use `QTGetComponentPropertyInfo` to discover the number of bytes you should allocate to hold this array.

`kQTSCAudioPropertyID_SampleRateRecommended = 'reco'`
: Clients not wishing to set an output sample rate manually may set the output rate to the recommended rate. Some compressors can perform rate conversion, and can pick optimal settings for a desired bitrate (AAC is one example). For other formats, the recommended rate is simply the closest output rate to the input rate that's allowed by the output format. `kQTSCAudioPropertyID_SampleRateIsRecommended` is read-only. To set the sample rate to recommended, a client sets the `kQTSCAudioPropertyID_BasicDescription` with `mSampleRate` = 0.0. To unset the sample rate as recommended, the client sets the `kQTSCAudioPropertyID_BasicDescription` with a non-zero `mSampleRate` field.

`kQTSCAudioPropertyID_ApplicableSampleRateList = 'avr#'`
: A read/listen C-style array of `AudioValueRange` values that specifies which of the value ranges in the `kQTSCAudioPropertyID_AvailableSampleRateList` are currently applicable. The `kQTSCAudioPropertyID_AvailableSampleRateList` takes into account client restrictions, and a compression format's general sample rate restrictions. `kQTSCAudioPropertyID_ApplicableSampleRateList` further filters the list to just those sample rates that are legal and valid given the current codec configuration. Use `QTGetComponentPropertyInfo` to discover the number of bytes you should allocate to hold the array.

`kQTSCAudioPropertyID_ClientRestrictedSampleRateList = 'crr#'`
: A read/write/listen C-style array of `AudioValueRange` values that specifies a client-restricted set of output sample rate ranges that should be listed as available. Use `QTGetComponentPropertyInfo` to discover the number of bytes you should allocate to hold this array.

`kQTSCAudioPropertyID_InputMagicCookie = 'ikki'`
: A read/write/listen opaque data structure that contains an untyped codec-specific data structure (a “magic cookie”), which some decompressors use to decode their input. Cookies are variable size, so you must call `QTGetComponentPropertyInfo` to discover the size of the buffer you should allocate to hold the cookie.

`kQTSCAudioPropertyID_MagicCookie = 'kuki'`
: A read/write/listen opaque data structure that contains an untyped codec-specific data structure (a “magic cookie”), which some decompressors use to configure their output. Cookies are variable size,

so you must call `QTGetComponentPropertyInfo` to discover the size of the buffer you should allocate to hold the cookie.

`kQTSCAudioPropertyID_ClientRestrictedLPCMBitsPerChannelList = 'crb#'`
: Specifies a client-restricted set of output bits per channel that should be listed as available. Use `QTGetComponentPropertyInfo` to discover the number of bytes you should allocate to hold the array.

`kQTSCAudioPropertyID_AvailableLPCMBitsPerChannelList = 'avb#'`
: A read/listen C-style array of `UInt32` values that contains a list of available bits per audio channel. This list is specific to LPCM, and takes into account any restrictions imposed by a client using the `kQTSCAudioPropertyID_LPCMBitsPerChannelList` property. Use `QTGetComponentPropertyInfo` to discover the number of bytes you should allocate to hold this array.

`kQTSCAudioPropertyID_ApplicableLPCMBitsPerChannelList = 'apb#'`
: Specifies which of the values in the `kQTSCAudioPropertyID_AvailableLPCMBitsPerChannelList` are currently applicable. The `kQTSCAudioPropertyID_AvailableLPCMBitsPerChannelList` takes into account client restrictions, and LPCM's general bits per channel restrictions. `kQTSCAudioPropertyID_ApplicableLPCMBitsPerChannelList` further filters the list to just those bits per channel that are legal and valid given the current LPCM configuration. Use `QTGetComponentPropertyInfo` to discover the number of bytes you should allocate to hold the array.

`kQTSCAudioPropertyID_LPCMBitsPerChannelList = 'sbc#'`
: A read/write/listen C-style array of `UInt32` values that contains a client-restricted set of output bits per channel, which you should list as available. Use `QTGetComponentPropertyInfo` to discover the number of bytes you should allocate to hold this array.

`kQTSCAudioPropertyID_AvailableNumChannelsList = 'anc#'`
: A read/listen C-style array of `UInt32` values that contains a list of available numbers of channels. This list is specific to the compression format and takes into account any restrictions imposed by a client using the `kQTSCAudioPropertyID_NumChannelsList` property. Use `QTGetComponentPropertyInfo` to discover the number of bytes you should allocate to hold this array.

`kQTSCAudioPropertyID_NumChannelsList = 'snc#'`
: A read/write/listen C-style array of `UInt32` values that contains a client-restricted set of numbers of channels that you should list as available. Use `QTGetComponentPropertyInfo` to discover the number of bytes you should allocate to hold this array.

`kQTSCAudioPropertyID_InputChannelLayout = 'icly'`
: A read/write/listen variable-size `AudioChannelLayout` structure that specifies the audio channel layout of the input description. `AudioChannelLayout` is a variable-size structure, so you must use `QTGetComponentPropertyInfo` to discover the number of bytes you should allocate for it.

`kQTSCAudioPropertyID_InputChannelLayoutName = 'icln'`
: A read-only `CFStringRef` structure that specifies the human-readable name for a `kQTSCAudioPropertyID_InputChannelLayout` structure, if one exists. The caller is responsible for calling `CFRelease` to dispose of the resulting string.

`kQTSCAudioPropertyID_ChannelLayout = 'clay'`
: A read/write/listen variable-size `AudioChannelLayout` structure that specifies the audio channel layout of the output description. `AudioChannelLayout` is a variable size structure, so you must use `QTGetComponentPropertyInfo` to discover the number of bytes you should allocate.

`kQTSCAudioPropertyID_ChannelLayoutName = 'clyn'`
: A read-only `CFStringRef` structure that specifies the human-readable name for a `kQTSCAudioPropertyID_ChannelLayout`, if one exists. The caller is responsible for calling `CFRelease` to dispose of the resulting string.

`kQTSCAudioPropertyID_ClientRestrictedChannelLayoutTagList = 'crl#'`
: Specifies a client-restricted set of channel layout tags that should be listed as available. Use `QTGetComponentPropertyInfo` to discover the number of bytes you should allocate to hold the array.

`kQTSCAudioPropertyID_AvailableChannelLayoutTagList = 'acl#'`
: A read/listen C-style array of `AudioChannelLayoutTag` values that specifies a list of available audio channel layout tags. This list is specific to the compression format and takes into account any restrictions imposed by a client using the `kQTSCAudioPropertyID_ChannelLayoutTagList` property. Use `QTGetComponentPropertyInfo` to discover the number of bytes you should allocate to hold this array.

`kQTSCAudioPropertyID_ChannelLayoutTagList = 'cly#'`
: A read/write C-style array of `AudioChannelLayoutTag` values that specifies a client-restricted set of channel layout tags, which you should list as available. Use `QTGetComponentPropertyInfo` to discover the number of bytes you should allocate to hold this array.

`kQTSCAudioPropertyID_AvailableChannelLayoutTagNamesList = 'cln#'`
: A read-only `CFArrayRef` array that specifies the human-readable names for the `AudioChannelLayoutTag` values in a `kQTSCAudioPropertyID_AvailableChannelLayoutTagList`. Each element in the array is a `CFStringRef` structure. The caller is responsible for calling `CFRelease` to dispose of this array.

`kQTSCAudioPropertyID_ApplicableChannelLayoutTagNamesList = 'apl#'`
: Specifies which of the values in the `kQTSCAudioPropertyID_AvailableChannelLayoutTagList` are currently applicable. The `kQTSCAudioPropertyID_AvailableChannelLayoutTagList` takes into account client restrictions, and the current output format's general channel layout restrictions. `kQTSCAudioPropertyID_ApplicableChannelLayoutTagList` further filters the list to just those channel layouts that are legal and valid given the current codec configuration. Use `QTGetComponentPropertyInfo` to discover the number of bytes you should allocate to hold the array.

`kQTSCAudioPropertyID_ClientRestrictedPCMFlags = 'crip'`
: Specifies a client-restricted set of flags corresponding to the `mFormatFlags` fields in an `AudioStreamBasicDescription`. Data type is a `SCAudioFormatFlagsRestrictions` struct. For instance, if a client wishes to specify to the StandardAudioCompression component that their file format requires little endian pcm data, the client may set this property, with `formatFlagsMask` set to `kAudioFormatFlagIsBigEndian`, and `formatFlagsValues` set to zero (indicating that the IsBigEndian bit should be interpreted as LittleEndian only).

`kQTSCAudioPropertyID_DiscreteChannelsOK = 'dscr'`
: A read/write/listen `Boolean` value that lets you tell the `StandardCompressionSubTypeAudio` dialog to not show "Discrete" as an available option. Each `AudioChannelLayout` structure assigns specific spatial orientation to specific channels (for example, Channel 1 = Left). "Discrete" is a special channel layout that does not assign spatial characteristics to channels, but instead labels them as distinct outputs. For example, the first channel in the audio source is played through the first channel on the output device, the second channel in the source is played through the second channel, and so on. If this property is set to `FALSE`, the `StandardCompressionSubTypeAudio` dialog will not show "Discrete" as an available option.

`kQTSCAudioPropertyID_LPCMSpecificFlagsMask = 'sffm'`
: A read/write/listen `UInt32` value that specifies which flag fields in `kQTSCAudioPropertyID_FormatSpecificFlags` should be made available in the `StandardCompressionSubTypeAudio` dialog. For instance, a value of `0xFFFFFFFD` (all bits except `kAudioFormatFlagIsBigEndian` set) tells the `StandardCompressionSubTypeAudio` component to disable any UI that would allow a choice between little and big endian. This selector is valid only for PCM formats and is ignored for others.

`kQTSCAudioPropertyID_InputSoundDescription = 'isdh'`
: A read/write `SoundDescriptionHandle` value that specifies the current input description as a `SoundDescriptionHandle` (lowest possible version for the current format). When calling `QTGetComponentProperty`, the caller passes a pointer to an unallocated `Handle` and assumes responsibility for calling `DisposeHandle` when done.

`kQTSCAudioPropertyID_SoundDescription = 'osdh'`
: A read/write `SoundDescriptionHandle` value that specifies the current output description as a `SoundDescriptionHandle` (lowest possible version for the current format). When calling `QTGetComponentProperty`, the caller passes a pointer to an unallocated `Handle` and assumes responsibility for calling `DisposeHandle` when done.

`kQTSCAudioPropertyID_InputBasicDescription = 'isbd'`
: A read/write/`DataProc`/listen `AudioStreamBasicDescription` value that specifies that the current input description is an `AudioStreamBasicDescription` value.

`kQTSCAudioPropertyID_BasicDescription = 'osbd'`
: A read/write/`DataProc`/listen `AudioStreamBasicDescription` value that specifies that the current output description is an `AudioStreamBasicDescription` value.

`kQTSCAudioPropertyID_CodecSpecificSettingsArray = 'cdst'`
: A read/write `CFArrayRef` structure that designates a `CFArray` of `CFDictionary` structures, which describe various parameters specific to configuring a codec. This array of dictionaries, which is published by some compressors, can be parsed to generate UI information. When any value in the array changes, a client should call `QTSetComponentProperty`, passing the entire array.

`kQTSCAudioPropertyID_SettingsState = scSettingsStateType`
: A read/write `Handle` value that is used to save the current state of the `StandardCompressionSubTypeAudio` component, so that its state may be restored at a later time with a single call. A `StandardCompressionSubTypeAudio` component can accept a saved settings state from a legacy `StandardCompressionSubTypeSound` component as write-only.

`kQTSCAudioPropertyID_ExtendedProcs = scExtendedProcsType`
: A read/write/listen `SCExtendedProcs` value that is used to get or set an `SCExtendedProcs` structure.

`kQTSCAudioPropertyID_PreferenceFlags = scPreferenceFlagsType`
: A read/write/listen `SInt32` value that is used to specify dialog preferences such as `scUseMovableModal`.

`kQTSCAudioPropertyID_WindowOptions = scWindowOptionsType`
: A read/write/listen `SCWindowSettings` structure that is used to set an `SCWindowSettings` structure, which tells the dialog about its parent window so that it can draw itself as a sheet on top of the parent.

Movie Exporter Properties

The following constants are used by movie export `getProperty` functions only (not `SCAudio`), so that variable size properties can be handled in that API where there is no associated size parameter. The `getProperty` function can be asked the size first, then the caller can allocate memory for the associated `SCAudio` property and call `getProperty` again to get the property.

```
enum {
  movieExportChannelLayoutSize  = 'clsz',               /* UInt32 */
  movieExportMagicCookieSize    = 'mcsz',               /* UInt32 */
  movieExportUseHighResolutionAudioProperties = 'hrau'   /* Boolean */
};
```

The `movieExportUseHighResolutionAudioProperties` constant is not a size. It is how the exporter asks a propertyProc if it is prepared to deal with high-res properties.

The `kPropertyClass_MovieExporter` constant defines the movie exporter class:

```
enum {
    kPropertyClass_MovieExporter = 'spit'
};
```

The `kMovieExporterPropertyID_EnableHighResolutionAudioFeatures` constant enables high-resolution audio features for `kPropertyClass_MovieExporter`. Its value is `Boolean`:

```
enum {
    kMovieExporterPropertyID_EnableHighResolutionAudioFeatures = 'hrau'
};
```

SGAudio Component Property Classes

Every `SGAudioMediaType` channel uses standard QuickTime component property selectors to get, set, and listen to properties. Each component property takes a property class as well as a property ID. `SGAudioMediaType` channels use the property classes listed in this section.

```
SGAudioMediaType                      = 'audi'
kQTPropertyClass_SGAudio                = 'audo'
kQTPropertyClass_SGAudioRecordDevice    = 'audr'
kQTPropertyClass_SGAudioPreviewDevice   = 'audp'
```

`'audo'`
: Used with properties that pertain to the `SGChannel` as a whole, or to the output of an `SGAudioChannel` (that is, with the resulting track in a QuickTime movie).

`'audr'`
: Used with properties that pertain specifically to an `SGAudioChannel` recording device's physical settings.

`'audp'`
: Used with properties that pertain specifically to an `SGAudioChannel` preview device's physical settings.

For the property IDs used with these classes. see "SGAudio Component Property IDs" (page 273).

SGAudio Component Property IDs

This section lists the property IDs for `SGAudioMediaType` channels. Besides the IDs defined below, `SGAudioMediaType` channels respond to `kComponentPropertyInfoList` and `kComponentPropertyClassPropertyInfo` selectors, which return `CFDataRef` structures containing arrays of `ComponentPropertyInfo` structures as defined in the file `ImageCompression.h`.

`kQTSGAudioPropertyID_DeviceListWithAttributes = '#dva'`
: Used with `kQTPropertyClass_SGAudio` in read and listen modes to get an array of `CFDictionaryRef` structures. Each dictionary represents the attributes of one audio device. See `Dictionary Keys` (page 281) for a list of supported dictionary keys. If the device list changes (for example, if a device is hotplugged or unplugged), listeners of this property will be notified. The caller is responsible for calling `CFRelease` to release the resulting `CFArray`.

`kQTSGAudioPropertyID_DeviceAttributes = 'deva'`
: Used with `kQTPropertyClass_SGAudioRecordDevice` and `kQTPropertyClass_SGAudioPreviewDevice` in read only mode to get a `CFDictionaryRef` structure representing the attributes of a specified audio device (record or preview). See `Dictionary Keys` (page 281) for a list of supported dictionary keys. Not all keys are guaranteed to be present for a given device. The caller is responsible for calling `CFRelease` to release the resulting `CFDictionary`.

`kQTSGAudioPropertyID_DeviceUID = 'uid ' [last character is space]`
: Used with `kQTPropertyClass_SGAudioRecordDevice` and `kQTPropertyClass_SGAudioPreviewDevice` in read and write modes to get a `CFString` with an audio device's unique ID for the current recording or preview or set the current recording or preview device to a specified audio device's unique ID. You can obtain a list of devices on the user's system with `kQTSGAudioPropertyID_DeviceListWithAttributes`. The caller is responsible for calling `CFRelease` to release the resulting `CFString`.

`kQTSGAudioPropertyID_ChannelLayout = 'clay'`
: Used with `kQTPropertyClass_SGAudioRecordDevice`, `kQTPropertyClass_SGAudio`, and `kQTPropertyClass_SGAudioPreviewDevice` in read and write modes to get or set an `AudioChannelLayout` structure representing the spatial or discrete channel layout. If used with `kQTPropertyClass_SGAudio`, the `AudioChannelLayout` refers to the channels in the resulting QuickTime movie sound track. If used with `kQTPropertyClass_SGAudioRecordDevice`, the `AudioChannelLayout` refers to the input channels on the record device. If used with `kQTPropertyClass_SGAudioPreviewDevice`, the `AudioChannelLayout` refers to the preview device's output channels. `AudioChannelLayout` is a variable size structure, so before calling `QTGetComponentProperty` you should call `QTGetComponentPropertyInfo` to discover the size of the block of memory you must allocate to hold the result.

`kQTSGAudioPropertyID_MagicCookie = 'kuki'`
: Used with `kQTPropertyClass_SGAudio` in read and write modes to access opaque data structures representing get or set compressor-specific out-of-band settings. This property is applicable only to compressed formats that use a cookie, such as AAC and AMR.

`kQTSGAudioPropertyID_ChannelMap = 'cmap'`
: Used with `kQTPropertyClass_SGAudioRecordDevice` in read and write modes to access a C-style array of `SInt32` structures that let a client enable or disable channels on a recording device, as well as reorder them or duplicate them to several output channels. This property need not be set if a client wishes to capture all channels from the record device; this is the default behavior. Each element in the `SInt32` array represents one output bus (into the `SGAudioChannel`) from the record device. The value of each element is the zero-based source channel on the input device that should feed the specified output. Channel-disabling example: if you wish to capture just the 1st, 3rd, and 5th channels from a 6-channel input device, your channel map should be `SInt32 map[3] = { 0, 2, 4 }`. Channel-reordering example: if you wish to capture both channels from a stereo input device, but you know the left and right channels are reversed in the data source, set your channel map to `SInt32 map[2] = { 1, 0 }`. Channel-duplication example: if you wish to duplicate the second source channel into 4 outputs, set your channel map to `SInt32 map[4] = { 1, 1, 1, 1 }`. Empty channel example: if you need to produce a conformant stream of audio (such as a 6-channel stream to send to an external 5.1 AC3 encoder), but you have audio only for the L, R, and C channels (on record device channels 0, 1, and 2), you may set your channel map to `SInt32 map[6] = { 0, 1, 2, -1, -1, -1 }`. The last 3 channels will be filled with silence.

`kQTSGAudioPropertyID_StreamFormat = 'frmt'`
: Used with `kQTPropertyClass_SGAudioRecordDevice`, `kQTPropertyClass_SGAudio`, and `kQTPropertyClass_SGAudioPreviewDevice` in read and write modes to access `AudioStreamBasicDescription` structures that let you get or set the format of the audio as it will be written to the destination QuickTime movie track. When used with `kQTPropertyClass_SGAudioRecordDevice`, this property ID gets and sets the format of audio as it is physically recorded on the device. The format must be one of the formats passed in `kQTSGAudioPropertyID_StreamFormatList`. The `mChannelsPerFrame` of the `StreamFormat` read from the record device will not reflect channels that have been enabled or disabled with the `ChannelMap` property.

`kQTSGAudioPropertyID_StreamFormatList = '#frm'`
: Used with `kQTPropertyClass_SGAudioRecordDevice` and `kQTPropertyClass_SGAudioPreviewDevice` in read-only mode to get an array of `AudioStreamBasicDescription` structures that describe valid combinations of settings supported by the physical device in its current configuration (sample rate, bit depth, number of channels).

`kQTSGAudioPropertyID_InputSelection = 'inpt'`
: Used with `kQTPropertyClass_SGAudioRecordDevice` in read and write modes to get an `OSType` value that lets you change the current input selection in devices that allow switching between data sources, such as analog, `adat`, `sdi`, `aes/ebu`, and `spdif`. When the input selection changes, the `StreamFormat` of the device may change as well; in particular, the number of channels may change.

`kQTSGAudioPropertyID_InputListWithAttributes = '#inp'`
: Used with `kQTPropertyClass_SGAudioRecordDevice` in read-only mode to get a `CFArrayRef` structure that represents the list of available input sources for a given device. A `CFArrayRef` of `CFDictionaryRef` values is returned, where each one represents the attributes of one input. See “`Dictionary Keys`” (page 281) for a list of valid keys. The caller is responsible for calling `CFRelease` to release the returned array.

`kQTSGAudioPropertyID_OutputSelection = 'otpt'`
: Used with `kQTPropertyClass_SGAudioPreviewDevice` in read and write modes to get an `OSType` value that lets you change the current output selection in devices that allow switching between output destinations, such as analog, `adat`, `sdi`, `aes/ebu`, and `spdif`. When the output selection changes, the `StreamFormat` of the device may change as well; in particular, the number of channels may change.

`kQTSGAudioPropertyID_OutputListWithAttributes = '#otp'`
: Used with `kQTPropertyClass_SGAudioPreviewDevice` in read-only mode to get a `CFArrayRef` structure that represents the list of available output destinations for a given device. A `CFArrayRef` of `CFDictionaryRef` values is returned, where each one represents the attributes of one output. See “`Dictionary Keys`” (page 281) for a list of valid keys. The caller is responsible for calling `CFRelease` to release the returned array.

`kQTSGAudioPropertyID_SoundDescription = 'snds'`
: Used with `kQTPropertyClass_SGAudio` in read and write modes to get a `SoundDescriptionHandle` value for the sound description that describes the data written to a QuickTime movie track. A `QTGetComponentProperty` call allocates the `SoundDescriptionHandle` for you. The caller should declare the `SoundDescriptionHandle` and set it to `NULL`, then pass its address to `QTGetComponentProperty`. The caller must call `DisposeHandle` to dispose of the resulting `SoundDescriptionHandle` when done with it.

`kQTSGAudioPropertyID_LevelMetersEnabled = 'lmet'`
: Used with `kQTPropertyClass_SGAudioRecordDevice`, `kQTPropertyClass_SGAudio`, and `kQTPropertyClass_SGAudioPreviewDevice` in read and write modes to access a `Boolean` value that controls metering. When used with `kQTPropertyClass_SGAudioRecordDevice` or `kQTPropertyClass_SGAudioPreviewDevice`, this property ID turns device level metering on or

off. When used with `kQTPropertyClass_SGAudio`, it turns output level metering on or off. When level meters are enabled, you can use `kQTSGAudioPropertyID_AveragePowerLevels` to get instantaneous levels, or `kQTSGAudioPropertyID_PeakHoldLevels` to get peak-hold style meters, which are better for clipping detection. Level meters should be enabled only if you intend to poll for levels, because they place an added load on the CPU when enabled.

`kQTSGAudioPropertyID_PeakHoldLevels = 'phlv'`
: Used with `kQTPropertyClass_SGAudioRecordDevice`, `kQTPropertyClass_SGAudio`, and `kQTPropertyClass_SGAudioPreviewDevice` in read-only mode to get a C-style array of `Float32` values representing in dB the peak hold levels for each channel on a device or output. This property ID may be used only when level meters are enabled (by using `kQTSGAudioPropertyID_LevelMetersEnabled`). Poll for peak hold levels as often as you would like, to update the user interface or look for clipping. The number of elements in the `Float32` array will be equal to the number of input channels on your record device for `kQTPropertyClass_SGAudioRecordDevice`, or the number of elements in your `kQTSGAudioPropertyID_ChannelMap`, if you've set one. It will be equal to the number of output channels on your preview device for `kQTPropertyClass_SGAudioPreviewDevice` and equal to the number of channels in your `kQTSGAudioPropertyID_StreamFormat` (`format.mChannelsPerFrame`) for `kQTPropertyClass_SGAudio`. If no channel mixdown is being performed between record device and output formats, then the `kQTSGAudioPropertyID_PeakHoldLevels` values for `kQTPropertyClass_SGAudioRecordDevice` and `kQTPropertyClass_SGAudio` will be equivalent. If you have requested hardware playthrough, level metering will be unavailable.

`kQTSGAudioPropertyID_AveragePowerLevels = 'aplv'`
: Used with `kQTPropertyClass_SGAudioRecordDevice`, `kQTPropertyClass_SGAudio`, and `kQTPropertyClass_SGAudioPreviewDevice` in read-only mode to get a C-style array of `Float32` values representing in dB the average power levels for each channel on a device or output. This property ID may be used only when level meters are enabled (by using `kQTSGAudioPropertyID_LevelMetersEnabled`). Poll for average power levels as often as you would like, to update the user interface. The number of elements in the `Float32` array will be equal to the number of input channels on your record device for `kQTPropertyClass_SGAudioRecordDevice`, or the number of elements in your `kQTSGAudioPropertyID_ChannelMap`, if you've set one. It will be equal to the number of output channels on your preview device for `kQTPropertyClass_SGAudioPreviewDevice` and equal to the number of channels in your `kQTSGAudioPropertyID_StreamFormat` (`format.mChannelsPerFrame`) for `kQTPropertyClass_SGAudio`. If no channel mixdown is being performed between record device and output formats, then the `kQTSGAudioPropertyID_AveragePowerLevels` values for `kQTPropertyClass_SGAudioRecordDevice` and `kQTPropertyClass_SGAudio` will be equivalent. If you have requested hardware playthrough, level metering will be unavailable.

`kQTSGAudioPropertyID_Settings = 'setu'`
: Used with `kQTPropertyClass_SGAudio` in read and write modes to access `UserData` values. This property takes supersedes the `SGGet/SetChannelSettings` calls. An `SGAudioMediaType` channel accepts old-style `'soun' SGChannel` settings in a `QTSetComponentProperty` call, but always produces new-style settings in a `QTGetComponentProperty` call.

`kQTSGAudioPropertyID_MasterGain = 'mgan'`
: Used with `kQTPropertyClass_SGAudioRecordDevice`, `kQTPropertyClass_SGAudio`, and `kQTPropertyClass_SGAudioPreviewDevice` in read and write modes to access a `Float32` value that represents the master gain on a physical recording device with 0.0 = minimum volume and 1.0 = the maximum volume of the device. With `kQTPropertyClass_SGAudioPreviewDevice`, this property gets or sets the master gain on the physical previewing device with 0.0 = minimum volume and 1.0 = the maximum volume of the device. With `kQTPropertyClass_SGAudio`, this property

gets or sets the master gain (volume) of the recorded audio data in software (pre-mixdown) with minimum = 0.0, maximum = unbounded. Normally you wouldn't set the volume greater than 1.0, but if the source sound level provided by the device is too low, you may set a gain greater than 1.0 to boost the gain. Some devices cannot respond to this property setting.

`kQTSGAudioPropertyID_PerChannelGain = 'cgan'`

Used with `kQTPropertyClass_SGAudioRecordDevice`, `kQTPropertyClass_SGAudio`, and `kQTPropertyClass_SGAudioPreviewDevice` in read and write modes to access a C-style array of `Float32` value that represents the gain of each channel on a physical recording device. The number of channels in the array for `kQTPropertyClass_SGAudioRecordDevice` and `kQTPropertyClass_SGAudioPreviewDevice` is equal to the total number of channels on the device, which can be discovered using `kQTSGAudioPropertyID_StreamFormat`. The number and order of channels in the array for the `kQTPropertyClass_SGAudio` class must correspond to the valence of channels on the output (which is affected by a channel map, if you've set one). With `kQTPropertyClass_SGAudio`, this property gets and sets the gain (volume) of each channel of recorded audio data in software. Levels set on the record device or preview device must be in the range minimum = 0.0, maximum = 1.0. Levels set in software may be set to values greater than 1.0 in order to boost low signals. The caller may specify that a particular channel gain level should be left alone by setting the value to –1.0. For instance, to set the gain of channels 1, 2, and 3 to 0.5 on a 6 channel device, pass the following array values in a `SetProperty` call: `{ 0.5, 0.5, 0.5, -1., -1., -1. }`.

`kQTSGAudioPropertyID_HardwarePlaythruEnabled = 'hard'`

Used with `kQTPropertyClass_SGAudioRecordDevice` in read and write modes to access a `Boolean` value representing the state of hardware playthrough during `seqGrabPreview` or `seqGrabPlayDuringRecord` operations. Setting this value will have no effect if the record device and preview device are not the same. Some devices do not support hardware playthrough; devices report whether or not they support this feature through the `kQTSGAudioPropertyID_DeviceListWithAttributes` property.

`kQTSGAudioPropertyID_ChunkSize = 'chnk'`

Used with `kQTPropertyClass_SGAudio` in read and write modes to access a `Float32` value representing the number of seconds of audio that the `SGAudioChannel` should buffer before writing.

`kComponentPropertyInfoList = 'list'`

Used with `kComponentPropertyClassPropertyInfo` in read-only mode as defined in the file `ImageCompression.h`.

`kQTSGAudioPropertyID_DeviceAlive = 'aliv'`

Used with `kQTPropertyClass_SGAudioRecordDevice` and `kQTPropertyClass_SGAudioPreviewDevice` in read and listen modes to get a `Boolean` value telling whether or not a device is alive. If the device is hot unplugged, listeners of this property will be notified. If a record or preview operation is in progress it will be stopped, but it is left to the client to select a new device.

`kQTSGAudioPropertyID_DeviceHogged = 'hogg'`

Used with `kQTPropertyClass_SGAudioRecordDevice` and `kQTPropertyClass_SGAudioPreviewDevice` in read, write, and listen modes to get a `Boolean` value telling whether a device has become hogged or unhogged by another process. If so, listeners of this property will be notified. `SGAudioMediaType` channel does not hog devices, but a client that has reason to gain exclusive access to a device may set this property to `TRUE`.

`kQTSGAudioPropertyID_DeviceInUse = 'used'`

Used with `kQTPropertyClass_SGAudioRecordDevice` and `kQTPropertyClass_SGAudioPreviewDevice` in read and listen modes to get a `Boolean` value that tells whether a device is in use. If the device starts to be used (for instance, when another process starts performing I/O with it), listeners of this property will be notified.

`kQTSGAudioPropertyID_MixerCoefficients = 'mixc'`
: Used with `kQTPropertyClass_SGAudio` in read and write modes to access a C-style array of `Float32` values representing a set of coefficients for mixdown. If you wish to perform a custom mixdown from the incoming record device channel valence (discoverable using a combination of `kQTPropertyClass_SGAudioRecordDevice`, `kQTSGAudioPropertyID_StreamFormat`, `kQTPropertyClass_SGAudioRecordDevice`, and `kQTSGAudioPropertyID_ChannelMap`) to a different output number of channels (using `kQTPropertyClass_SGAudio` and `kQTSGAudioPropertyID_StreamFormat`), you may specify your own set of mixer coefficients which will be set as volume values at each crosspoint in `SGAudioMediaType`'s internal matrix mixer. The value you pass is a two-dimensional array of `Float32` values where the first dimension (rows) is the input channel and the second dimension (columns) is the output channel. Each `Float32` value contains one gain level to apply.

`kQTSGAudioPropertyID_PreMixCallback = '_mxc'`
: Used with `kQTPropertyClass_SGAudio` in read and write modes to access a pre-mix `SGAudioCallbackStruct` (page 261). If you wish to receive a callback when new audio samples become available from a recording device (before they've been mixed down), set this property using an `SGAudioCallbackStruct` containing a pointer to your `SGAudioCallback` function and a reference constant (*`RefCon`*). If you have previously registered a callback and no longer wish to receive it, call `QTSetComponentProperty` again, this time passing `NULL` for your *`inputProc`* and 0 for your *`inputRefCon`*.

`kQTSGAudioPropertyID_PreMixCallbackFormat = '_mcf'`
: Used with `kQTPropertyClass_SGAudio` in read-only mode to get an `AudioStreamBasicDescription` structure representing the format of the audio that will be received by your pre-mix `SGAudioCallback` function. Note that the format may not be available until you've called `SGPrepare`.

`kQTSGAudioPropertyID_PostMixCallback = 'mx_c'`
: Used with `kQTPropertyClass_SGAudio` in read and write modes to access a post-mix `SGAudioCallbackStruct` (page 261). If you wish to receive a callback after audio samples have been mixed (the first step after they are received from a recording device by `SGAudioMediaType` channel), set this property ID using an `SGAudioCallbackStruct` containing a pointer to your `SGAudioCallback` function and a reference constant (*`RefCon`*). If you have previously registered a callback and no longer wish to receive it, call `QTSetComponentProperty` again, this time passing `NULL` for your *`inputProc`* and 0 for your *`inputRefCon`*.

`kQTSGAudioPropertyID_PostMixCallbackFormat = 'm_cf'`
: Used with `kQTPropertyClass_SGAudio` in read-only mode to get an `AudioStreamBasicDescription` structure representing the format of the audio that will be received by your post-mix `SGAudioCallback` function. Note that the format may not be available until you've called `SGPrepare`.

`kQTSGAudioPropertyID_PreConversionCallback = '_cvc'`
: Used with `kQTPropertyClass_SGAudio` in read and write modes to access a pre-conversion `SGAudioCallbackStruct` (page 261). If you wish to receive a callback just before audio samples are about to be sent through an audio converter (for format conversion or compression), set this property ID using an `SGAudioCallbackStruct` containing a pointer to your `SGAudioCallback` function and a reference constant (*`RefCon`*). If you have previously registered a callback and no longer wish to receive it, call `QTSetComponentProperty` again, this time passing `NULL` for your *`inputProc`* and 0 for your *`inputRefCon`*.

`kQTSGAudioPropertyID_PreConversionCallbackFormat = '_ccf'`
: Used with `kQTPropertyClass_SGAudio` in read-only mode to get an `AudioStreamBasicDescription` structure representing the format of the audio that will be received

by your pre-conversion `SGAudioCallback` function. Note that the format may not be available until you've called `SGPrepare`.

`kQTSGAudioPropertyID_PostConversionCallback = 'cv_c'`
: Used with `kQTPropertyClass_SGAudio` in read and write modes to access a post-conversion `SGAudioCallbackStruct` (page 261). If you wish to receive a callback right after audio samples have been sent through an audio converter (for format conversion or compression), set this property ID using an `SGAudioCallbackStruct` containing a pointer to your `SGAudioCallback` function and a reference constant (*`RefCon`*). If you have previously registered a callback and no longer wish to receive it, call `QTSetComponentProperty` again, this time passing `NULL` for your *`inputProc`* and 0 for your *`inputRefCon`*.

`kQTSGAudioPropertyID_PostConversionCallbackFormat = 'c_cf'`
: Used with `kQTPropertyClass_SGAudio` in read-only mode to get an `AudioStreamBasicDescription` structure representing the format of the audio that will be received by your post-conversion `SGAudioCallback` function. Note that the format may not be available until you've called `SGPrepare`.

Sound Description Property IDs

The following constants identify sound description properties.

```
enum {
    kQTSoundDescriptionPropertyID_AudioChannelLayout = 'clay',
    kQTSoundDescriptionPropertyID_MagicCookie = 'kuki',
    kQTSoundDescriptionPropertyID_AudioStreamBasicDescription = 'asbd',
    kQTSoundDescriptionPropertyID_UserReadableText = 'text'
};
```

`kQTSoundDescriptionPropertyID_AudioChannelLayout = 'clay'`
: Used to get or set an `AudioChannelLayout` value. This is a variable-size property because it may contain an array of Channel Descriptions. You must get the size by calling `QTSoundDescriptionGetPropertyInfo`, allocate a structure of that size, and then get the property.

`kQTSoundDescriptionPropertyID_MagicCookie = 'kuki'`
: Used to get or set opaque bytes. This is a variable-size property, because it is completely defined by the codec that uses the cookie. You must get the size by calling `QTSoundDescriptionGetPropertyInfo`, allocate a structure of that size, and then get the property.

`kQTSoundDescriptionPropertyID_AudioStreamBasicDescription = 'asbd'`
: Used to get an `AudioStreamBasicDescription` value.

`kQTSoundDescriptionPropertyID_UserReadableText = 'text'`
: Used to get a `CFStringRef` value. `QTSoundDescriptionGetProperty` does a `CFRetain` of the returned `CFString` on behalf of the caller, so the caller is responsible for calling `CFRelease` on the returned `CFString`.

Audio Property IDs

The following constants identify audio properties.

```
enum {
    kQTAudioPropertyID_Gain = 'gain',
    kQTAudioPropertyID_Mute = 'mute',
    kQTAudioPropertyID_Balance = 'bala',
    kQTAudioPropertyID_Fade = 'fade',
    kQTAudioPropertyID_SummaryChannelLayout = 'clay',
    kQTAudioPropertyID_DeviceChannelLayout = 'dcly',
    kQTAudioPropertyID_FormatString = 'fstr',
    kQTAudioPropertyID_ChannelLayoutString = 'lstr',
    kQTAudioPropertyID_SampleRateString = 'rstr',
    kQTAudioPropertyID_SampleSizeString = 'sstr',
    kQTAudioPropertyID_BitRateString = 'bstr',
    kQTAudioPropertyID_SummaryString = 'asum'
};
```

`kQTAudioPropertyID_Gain = 'gain'`
: Used to get and set a `Float32` value that represents the audio gain of a movie or track. The gain level is multiplicative; eg. 0.0 is silent, 0.5 is –6dB, 1.0 is 0dB (ie. the audio from the movie is not modified), 2.0 is +6dB, etc. The gain level can be set higher than 1.0 in order to allow quiet movies and tracks to be boosted in volume. Settings higher than 1.0 may result in audio clipping, of course. The setting is not stored in the movie or track; it is used only until the movie or track is disposed.

`kQTAudioPropertyID_Mute = 'mute'`
: Used to get and set a `Boolean` value that indicates the audio mute state of a movie or track. If TRUE, the movie or track is muted. The setting is not stored in the movie or track; it is used only until the movie or track is disposed.

`kQTAudioPropertyID_Balance = 'bala'`
: Used to get and set a `Float32` value that represents the audio balance of a movie. It is supported only for movies, not tracks. –1.0 means full left, 0.0 means centered, and 1.0 means full right. The setting is not stored in the movie; it is used only until the movie is disposed.

`kQTAudioPropertyID_Fade = 'fade'`
: Used to get and set a `Float32` value that represents the audio fade of a movie. It is supported only for movies, not tracks. 1.0 means full forward, 0.0 means centered, and –1.0 means full rearward. The setting is not stored in the movie; it is used only until the movie is disposed.

`kQTAudioPropertyID_SummaryChannelLayout = 'clay'`
: Used to get an `AudioChannelLayout` value that represents the summary audio channel layout of a movie or other grouping of audio streams. All like-labelled channels are combined, so there are no duplicates. For example, if there is a stereo (L/R) track, 5 single-channel tracks marked Left, Right, Left Surround, Right Surround and Center, and a 4 channel track marked L/R/Ls/Rs, then the summary AudioChannelLayout will be L/R/Ls/Rs/C—It will _not_ be L/R/L/R/Ls/Rs/C/L/R/Ls/Rs. This is a variable-size property, because it it may contain an array of channel descriptions. You must get the size by calling a function such as `QTGetMoviePropertyInfo`, allocate a structure of that size, and then get the property.

`kQTAudioPropertyID_DeviceChannelLayout = 'dcly'`
: Used to get an `AudioChannelLayout` value that represents the audio channel layout of the device a movie is playing to. This is a variable-size property, because it it may contain an array of channel descriptions. You must get the size by calling a function such as `QTGetMoviePropertyInfo`, allocate a structure of that size, and then get the property.

`kQTAudioPropertyID_FormatString = 'fstr'`
: Used with `kQTPropertyClass_Audio` to get a `CFStringRef` value containing a localized, human readable string that describes an audio format; for example, "MPEG Layer 3." You may get this property from a `SoundDescription` handle by calling `QTSoundDescriptionGetProperty` or from a `StandardAudioCompression` (`scdi` or `audi`) component instance by calling `QTGetComponentProperty`.

`kQTAudioPropertyID_ChannelLayoutString = 'lstr'`
: Used with `kQTPropertyClass_Audio` to get a `CFStringRef` value containing a localized, human readable string that describes an audio channel layout; for example, "5.0 (L R C Ls Rs)." You may get

this property from a `SoundDescription` handle by calling `QTSoundDescriptionGetProperty` or from a `StandardAudioCompression` (`scdi` or `audi`) component instance by calling `QTGetComponentProperty`.

`kQTAudioPropertyID_SampleRateString = 'rstr'`
: Used to get a `CFStringRef` value containing a localized, human readable string that describes an audio sample rate; for example, "44.100 kHz." You may get this property from a `SoundDescription` handle by calling `QTSoundDescriptionGetProperty` or from a `StandardAudioCompression` (`scdi` or `audi`) component instance by calling `QTGetComponentProperty`.

`kQTAudioPropertyID_SampleSizeString = 'sstr'`
: Used to get a `CFStringRef` value containing a localized, human readable string that describes an audio sample size; for example, "24-bit." This property will return a valid string only if the audio format is uncompressed (LPCM). You may get this property from a `SoundDescription` handle by calling `QTSoundDescriptionGetProperty` or from a `StandardAudioCompression` (`scdi` or `audi`) component instance by calling `QTGetComponentProperty`.

`kQTAudioPropertyID_BitRateString = 'bstr'`
: Used to get a `CFStringRef` value containing a localized, human readable string that describes an audio bit rate; for example, "12 kbps." You may get this property from a `StandardAudioCompression` (`scdi` or `audi`) component instance by calling `QTGetComponentProperty`.

`kQTAudioPropertyID_SummaryString = 'asum'`
: Used to get a `CFStringRef` value containing a localized, human readable string that summarizes an audio format; for example, "16-bit Integer (Big Endian), Stereo (L R), 48.000 kHz." You may get this property from a `SoundDescription` handle by calling `QTSoundDescriptionGetProperty` or from a `StandardAudioCompression` (`scdi` or `audi`) component instance by calling `QTGetComponentProperty`.

Dictionary Keys

The dictionary keys listed in this section are used with c ertain of the property IDs listed in "`SGAudio Component Property IDs`" (page 273). They may be used to parse CF dictionaries returned by `kQTSGAudioPropertyID_DeviceListWithAttributes` and `kQTSGAudioPropertyID_DeviceAttributes` IDs for `SGAudioMediaType` channels.

`kQTAudioDeviceAttribute_DeviceUIDKey = 'uid '`
: A `CFStringRef` containing a unique identifier for a device.

`kQTAudioDeviceAttribute_DeviceNameKey = 'name'`
: A `CFStringRef` containing a device's printable name, suitable for the user interface.

`kQTAudioDeviceAttribute_DeviceManufacturerKey = 'manu'`
: A `CFStringRef` containing a device manufacturer's printable name, suitable for the user interface.

`kQTAudioDeviceAttribute_DeviceTransportTypeKey = 'tran'`
: A `CFNumberRef` that wraps an `OSType`; for example, `'1394'` for `fw`. See the file `IOAudioTypes.h`.

`kQTAudioDeviceAttribute_DeviceAliveKey = 'aliv'`
: A `CFBooleanRef` value that is `TRUE` if the device is present.

`kQTAudioDeviceAttribute_DeviceCanRecordKey = 'rec ' [last char = space]`
: A `CFBooleanRef` value that is `TRUE` if the device can be used for recording (some devices can only play back).

`kQTAudioDeviceAttribute_DeviceCanPreviewKey = 'prev'`
: A `CFBooleanRef` value that is `TRUE` if the device can be used to preview a grabbed sequence.

`kQTAudioDeviceAttribute_DeviceHoggedKey = 'hogg'`
: A `CFNumberRef` that wraps the unique process ID that is hogging the device, or –1 if the device is currently not being hogged. The process ID comes from a call to `getpid`.

`kQTAudioDeviceAttribute_DeviceInUseKey = 'used'`
: A `CFBooleanRef` value that is `TRUE` if the device is performing I/O in any process.

`kQTAudioDeviceAttribute_DeviceSupportsHardwarePlaythruKey = 'hard'`
: A `CFBooleanRef` value that is `TRUE` if the device supports hardware playthrough of inputs to outputs.

`kQTAudioDeviceAttribute_DefaultInputDeviceKey = 'dIn ' [last char = space]`
: A `CFBooleanRef` value that's `TRUE` if the device is the user-selected default input in an audio MIDI setup.

`kQTAudioDeviceAttribute_DefaultOutputDeviceKey = 'dOut'`
: A `CFBooleanRef` value that's `TRUE` if the device is the user-selected default output in an audio MIDI setup.

`kQTAudioDeviceAttribute_DefaultSystemOutputDeviceKey = 'sOut'`
: A `CFBooleanRef` value that's `TRUE` if the device is the user-selected device where system alerts play.

`kQTAudioDeviceAttribute_IsCoreAudioDeviceKey = 'hal!'`
: A `CFBooleanRef` value that's `TRUE` if the device is a Core Audio device.

Device Attribute Keys for Inputs and Outputs

The following dictionary keys may be used to parse CF dictionaries returned by `kQTSGAudioPropertyID_DeviceListWithAttributes` and `kQTSGAudioPropertyID_DeviceAttributes` IDs for `SGAudioMediaType` channels.

`kQTAudioDeviceAttribute_DeviceInputID = 'inID'`
: A `CFNumberRef` that wraps an `OSType` value.

`kQTAudioDeviceAttribute_DeviceInputDescription = 'inds'`
: A `CFStringRef` that is suitable for displaying to the user.

`kQTAudioDeviceAttribute_DeviceOutputID = 'otID'`
: A `CFNumberRef` that wraps an `OSType` value.

`kQTAudioDeviceAttribute_DeviceOutputDescription = 'otds'`
: A `CFStringRef` that is suitable for displaying to the user.

Sequence Grabber Setting Codes

The following setting codes are used by sequence grabber channels of type `SGAudioMediaType`.

```
enum {
  sgcAudioRecordDeviceSettingsAtom  = kQTPropertyClass_SGAudioRecordDevice,
  sgcAudioPreviewDeviceSettingsAtom = kQTPropertyClass_SGAudioPreviewDevice,
  sgcAudioOutputSettingsAtom        = kQTPropertyClass_SGAudio,
  sgcAudioSettingsVersion           = 'vers',
  sgcAudioDeviceUID                 = kQTAudioDeviceAttribute_DeviceUIDKey,
  sgcAudioDeviceName                = kQTAudioDeviceAttribute_DeviceNameKey,
  sgcAudioStreamFormat              = kQTSGAudioPropertyID_StreamFormat,
  sgcAudioInputSelection            = kQTSGAudioPropertyID_InputSelection,
  sgcAudioOutputSelection           = kQTSGAudioPropertyID_OutputSelection,
  sgcAudioChannelMap                = kQTSGAudioPropertyID_ChannelMap,
  sgcAudioMasterGain                = kQTSGAudioPropertyID_MasterGain,
  sgcAudioPerChannelGain            = kQTSGAudioPropertyID_PerChannelGain,
  sgcAudioLevelMetersEnabled      =    kQTSGAudioPropertyID_LevelMetersEnabled,
  sgcAudioChannelLayout             = kQTSGAudioPropertyID_ChannelLayout,
  sgcAudioMixerCoefficients         = kQTSGAudioPropertyID_MixerCoefficients,
  sgcAudioMagicCookie               = kQTSGAudioPropertyID_MagicCookie
};
```

Metadata Format Constants

The format constants in this section are used with functions of the form `QTMetaData`...

Following are constants for the `QTMetaDataStorageFormat` type:

`kQTMetaDataStorageFormatQuickTime = 'mdta'`
: The QuickTime metadata storage format

`kQTMetaDataKeyFormatQuickTime = 'mdta'`
: Reverse DNS format

`kQTMetaDataStorageFormatiTunes = 'itms'`
: The iTunes metadata storage format

Following are constants for the `QTMetaDataKeyFormat` type:

`kQTMetaDataKeyFormatiTunesShortForm = 'itsk'`
: A four-character code

`kQTMetaDataKeyFormatiTunesLongForm = 'itlk'`
: Reverse DNS format

Following are constants for user data formats:

`kQTMetaDataStorageFormatUserData = 'udta'`
: User data storage format

`kQTMetaDataKeyFormatUserData = 'udta',`
: User data key storage format

Metadata Property IDs

The property IDs in this section are used with functions of the form `QTMetaData`...

Following are constants for the `QTMetaDataRef` type:

`kPropertyClass_QTMetaData = 'meta'`
: The QuickTime metadata property class.

`kQTMetaDataPropertyID_StorageFormats = 'fmts'`
: The list of storage formats of type `QTMetaDataStorageFormat` associated with a `QTMetaDataRef` object. The read-only return value is a C-style array of `OSType` values.

`kQTMetaDataPropertyID_OwnerType = 'ownt'`
: The owner type associated with a `QTMetaDataRef` object. The read-only return value is an `OSType` (`QT_MOVIE_TYPE`, `QT_TRACK_TYPE`, or `QT_MEDIA_TYPE`).

`kQTMetaDataPropertyID_Owner = 'ownr'`
: The owner associated with a `QTMetaDataRef` object, which does not necessarily need an owner. The read-only return value is type `Movie`, `Track`, or `Media`.

Following are constants for the `QTMetaDataItem` type:

`kPropertyClass_QTMetaDataItem = 'mdit'`
: The metadata item property class ID

`kQTMetaDataItemPropertyID_Value = 'valu'`
: The value of the metadata item. The read-only return value is a C-style array of values of type `UInt8`.

`kQTMetaDataItemPropertyID_DataType = 'dtyp'`
: The value type of the metadata item. The read/write return value is type `UInt32`.

`kQTMetaDataItemPropertyID_StorageFormat = 'sfmt'`
: The storage format of the metadata item. The read-only return value is type `QTMetaDataStorageFormat`.

`kQTMetaDataItemPropertyID_Key = 'key '` [last char is space]
: The key associated with the metadata item. The read/write return value is a C-style array of values of type `UInt8`.

`kQTMetaDataItemPropertyID_KeyFormat = 'keyf'`
: The format of the metadata item key. The read/write return value is type `OSType`.

`kQTMetaDataItemPropertyID_Locale = 'loc '`
: The locale identifier based on the naming convention defined by the International Components for Unicode (ICU). The identifier consists of two pieces of ordered information: a language code and a region code. The language code is based on the ISO 639-1 standard, which defines two-character codes, such as `en` and `fr`, for the world's most commonly used languages. If a two-letter code is not available, then ISO 639-2 three-letter identifiers are accepted as well; for example, `haw` for Hawaiian. The region code is defined by ISO 3166-1. It is all uppercase and is appended, with an underscore, after the language code; for example `en_US`, `en_GB`, and `fr_FR`. The read/write return value is a C string of type `UInt32`.

Metadata Key Constants

The following key constants are used with functions of the form `QTMetaData`...

```
// Pre-defined common keys
    kQTMetaDataCommonKeyAuthor          = 'auth'
    kQTMetaDataCommonKeyComment         = 'cmmt'
    kQTMetaDataCommonKeyCopyright       = 'cprt'
    kQTMetaDataCommonKeyDirector        = 'dtor'
    kQTMetaDataCommonKeyDisplayName     = 'name'
    kQTMetaDataCommonKeyInformation     = 'info'
    kQTMetaDataCommonKeyKeywords        = 'keyw'
    kQTMetaDataCommonKeyProducer        = 'prod'

// Mapping from common keys to user data identifiers:
    kQTMetaDataCommonKeyAuthor          -> kUserDataTextAuthor
    kQTMetaDataCommonKeyComment         -> kUserDataTextComment
    kQTMetaDataCommonKeyCopyright       -> kUserDataTextCopyright
    kQTMetaDataCommonKeyDirector        -> kUserDataTextDirector
    kQTMetaDataCommonKeyDisplayName     -> kUserDataTextFullName
    kQTMetaDataCommonKeyInformation     -> kUserDataTextInformation
    kQTMetaDataCommonKeyKeywords        -> kUserDataTextKeywords
    kQTMetaDataCommonKeyProducer        -> kUserDataTextProducer
```

Metadata Error Codes

The following error codes are returned by functions of the form `QTMetaData`...

```
kQTMetaDataInvalidMetaDataErr         = -2173
kQTMetaDataInvalidItemErr             = -2174
kQTMetaDataInvalidStorageFormatErr    = -2175
kQTMetaDataInvalidKeyFormatErr        = -2176
kQTMetaDataNoMoreItemsErr             = -2177
```

New Movie Property Codes

The following codes are stored in the *propClass* fields of `QTNewMoviePropertyElement` (page 260) data structures, which pass them to `NewMovieFromProperties` (page 191).

```
kQTPropertyClass_DataLocation = 'dloc',
        kQTDataLocationPropertyID_DataReference             = 'dref',
           // DataReferenceRecord *
        kQTDataLocationPropertyID_CFStringNativePath        = 'cfnp',
           // CFStringRef *
        kQTDataLocationPropertyID_CFStringPosixPath         = 'cfpp',
           // CFStringRef *
        kQTDataLocationPropertyID_CFStringHFSPath           = 'cfhp',
           // CFStringRef *
        kQTDataLocationPropertyID_CFStringWindowsPath       = 'cfwp',
           // CFStringRef *
        kQTDataLocationPropertyID_CFURL                     = 'cfur',
           // CFURLRef *
        kQTDataLocationPropertyID_QTDataHandler             = 'qtdh',
           // DataHandler *
        kQTDataLocationPropertyID_Scrap                     = 'scrp',
           // NULL
        kQTDataLocationPropertyID_LegacyMovieResourceHandle = 'rezh',
           // Handle *
        kQTDataLocationPropertyID_MovieUserProc             = 'uspr',
           // QTNewMovieUserProcRecord *
        kQTDataLocationPropertyID_ResourceFork              = 'rfrk',
           // SInt16 *
        kQTDataLocationPropertyID_DataFork                  = 'dfrk',
           // SInt16 *

 kQTPropertyClass_Context = 'ctxt',
        kQTContextPropertyID_AudioContext       = 'audi',
           // QTAudioContextRef *
        kQTContextPropertyID_VisualContext      = 'visu',
           // QTVisualContextRef *

 kQTPropertyClass_MovieResourceLocator = 'rloc',
        kQTMovieResourceLocatorPropertyID_LegacyResID   = 'rezi',
           // SInt16 * (input/output property)
        kQTMovieResourceLocatorPropertyID_LegacyResName = 'rezn',
           // Str255   (output property)
        kQTMovieResourceLocatorPropertyID_FileOffset    = 'foff',
           // UInt64 *
        kQTMovieResourceLocatorPropertyID_Callback      = 'calb',
           // User-defined
```

```
kQTPropertyClass_MovieInstantiation = 'mins',
       kQTMovieInstantiationPropertyID_DontResolveDataRefs          = 'rdrn',
          // Boolean *
       kQTMovieInstantiationPropertyID_DontAskUnresolvedDataRefs    = 'aurn',
          // Boolean *
       kQTMovieInstantiationPropertyID_DontAutoAlternates           = 'aaln',
          // Boolean *
       kQTMovieInstantiationPropertyID_DontUpdateForeBackPointers   = 'fbpn',
          // Boolean *
       kQTMovieInstantiationPropertyID_AsyncOK                      = 'asok',
          // Boolean *
       kQTMovieInstantiationPropertyID_IdleImportOK                 = 'imok',
          // Boolean *
       kQTMovieInstantiationPropertyID_DontAutoUpdateClock          = 'aucl',
          // Boolean *
       kQTMovieInstantiationPropertyID_ResultDataLocationChanged    = 'dlch',
          // Boolean * (output property)

kQTPropertyClass_NewMovieProperty = 'mprp',
       kQTNewMoviePropertyID_DefaultDataRef        = 'ddrf',
          // DataReferenceRecord *
       kQTNewMoviePropertyID_Active                = 'actv',
          // Boolean *
       kQTNewMoviePropertyID_DontInteractWithUser  = 'intn',
          // Boolean *
```

Document Revision History

This table describes the changes to *QuickTime 7 Update Guide*.

Date	Notes
2005-04-29	Updated for public release of QuickTime 7. Changed title from "What's New in QuickTime 6.6 Developer Preview."