QuickTime Framework Reference

QuickTime

2006-05-23

Contents

Introduction **Introduction to Quicktime Framework Reference 9**

Part I **Managers 11**

Chapter 1 **Image Compression Manager Reference 13**

Overview 13
Functions by Task 13
Functions 21
Callbacks 130
Data Types 136
Constants 140

Chapter 2 **Movie Manager Reference 163**

Overview 163
Functions by Task 163
Functions 177
Callbacks 351
Data Types 351
Constants 356

Part II **Constants 361**

Chapter 3 **QuickTime Constants Reference 363**

Overview 363
Constants 363

Part III **Other References 443**

Chapter 4 **Component Creation Reference for QuickTime 445**

Overview 445
Functions by Task 445
Functions 453
Callbacks 551
Data Types 555
Constants 565

Chapter 5 Compression and Decompression Reference for QuickTime 575

Overview 575
Functions by Task 575
Functions 586
Callbacks 740
Data Types 740
Constants 748

Chapter 6 Data Components Reference for QuickTime 753

Overview 753
Functions by Task 753
Functions 759
Callbacks 842
Data Types 842
Constants 846

Chapter 7 Image Codec Reference for QuickTime 849

Overview 849
Functions by Task 849
Functions 854
Callbacks 932
Data Types 933
Constants 957

Chapter 8 Import and Export Reference for QuickTime 965

Overview 965
Functions by Task 965
Functions 974
Callbacks 1080
Data Types 1081
Constants 1081

Chapter 9 Media Types and Media Handlers Reference 1087

Overview 1087
Functions by Task 1087
Functions 1093
Callbacks 1164
Data Types 1164
Constants 1170

Chapter 10 Movie Controller Reference 1175

Overview 1175
Functions by Task 1175
Functions 1179
Callbacks 1223
Data Types 1224
Constants 1225

Chapter 11 QuickTime Movie Properties Reference 1229

Overview 1229
Functions by Task 1229
Functions 1232
Callbacks 1276
Data Types 1276
Constants 1278

Chapter 12 Movie Toolkit Reference 1283

Overview 1283
Functions by Task 1283
Functions 1298
Callbacks 1504
Data Types 1509
Constants 1516

Chapter 13 QuickTime Movie Track and Media Reference 1525

Overview 1525
Functions by Task 1525
Functions 1534
Callbacks 1661
Data Types 1661
Constants 1665

Chapter 14 QuickTime Music Architecture Reference 1669

Overview 1669
Functions by Task 1669
Functions 1676
Callbacks 1761
Data Types 1763
Constants 1777

Chapter 15 QuickTime Streaming Reference 1791

Overview 1791
Functions 1791
Callbacks 1931
Data Types 1932
Constants 1957

Chapter 16 QuickTime Virtual Reality Reference 1969

Overview 1969
Functions by Task 1969
Functions 1975
Callbacks 2065
Data Types 2069
Constants 2075

Chapter 17 Sequence Grabber Reference for QuickTime 2085

Overview 2085
Functions by Task 2085
Functions 2096
Callbacks 2228
Data Types 2236
Constants 2245

Chapter 18 Video Components Reference for QuickTime 2251

Overview 2251
Functions by Task 2251
Functions 2258
Callbacks 2340
Data Types 2340
Constants 2349

Chapter 19 Windows API Reference for QuickTime 2361

Overview 2361
Functions 2361
Callbacks 2365
Data Types 2365
Constants 2365

Chapter 20 QuickTime Atoms and Resources Reference 2367

Overview 2367

Chapter 21 **QuickTime Atoms 2369**

Atoms 2369

Chapter 22 **QuickTime Public Resources 2455**

Resources 2455

Document Revision History 2477

Index 2479

Introduction to Quicktime Framework Reference

Declared in	Components.h Controls.h Dialogs.h Files.h HIMovieView.h HIObject.h IOHIDDescriptorParser.h IOMacOSTypes.h IOMacOSVideo.h ImageCodec.h ImageCompression.h MacErrors.h MacTypes.h MediaHandlers.h Movies.h OSTypes.h OSUtils.h QTML.h QTSMovie.h QTStreamingComponents.h QuickTimeComponents.h QuickTimeMusic.h QuickTimeStreaming.h QuickTimeVR.h QuickdrawTypes.h Script.h Sound.h TextEdit.h

This collection of documents provides the API reference for the QuickTime framework. This framework provides C functions to support creation and display of multimedia in the form of QuickTime movies for both Mac OS and Windows. The functions in this reference collection are organized into reference documents according to the header file in which they are declared.

Note: The Objective-C API for QuickTime is documented in the *QTKit Framework Reference*.

Managers

Image Compression Manager Reference

Framework:	Frameworks/QuickTime.framework
Declared in	ImageCompression.h

Overview

Applications can use the QuickTime image compression APIs to compress and decompress sounds, images, and image sequences, as well as to transcode sounds and images between compression formats.

Functions by Task

Image Transcoder Support

`ImageTranscoderBeginSequence` (page 103)
Initiates an image transcoding sequence and specifies the input data format.

`ImageTranscoderConvert` (page 104)
Performs image transcoding operations.

`ImageTranscoderDisposeData` (page 105)
Disposes of transcoded data.

`ImageTranscoderEndSequence` (page 106)
Ends an image transcoding sequence.

Managing an ICM Compression Session

`ICMCompressionSessionCompleteFrames` (page 34)
Forces a compression session to complete encoding frames.

`ICMCompressionSessionCreate` (page 35)
Creates a compression session for a specified codec type.

`ICMCompressionSessionEncodeFrame` (page 37)
Presents video frames to a compression session.

`ICMCompressionSessionGetImageDescription` (page 38)
Retrieves the image description for a video compression session.

`ICMCompressionSessionGetPixelBufferPool` (page 39)
Returns a pool that can provide ideal source pixel buffers for a compression session.

`ICMCompressionSessionGetProperty` (page 40)
Retrieves the value of a specific property of a compression session.

Using the OpenGL Texture Context

`QTOpenGLTextureContextCreate` (page 121)
Creates a new OpenGL texture context for a specified OpenGL context and pixel format.

`QTVisualContextCopyImageForTime` (page 125)
Retrieves an image buffer from the visual context, indexed by the provided time.

`QTVisualContextGetAttribute` (page 125)
Returns a visual context attribute.

`QTVisualContextGetTypeID` (page 126)
Returns the CFTypeID for QTVisualContextRef.

`QTVisualContextIsNewImageAvailable` (page 126)
Queries whether a new image is available for a given time.

`QTVisualContextRelease` (page 127)
Releases a visual context object.

`QTVisualContextRetain` (page 128)
Retains a visual context object.

`QTVisualContextSetAttribute` (page 128)
Sets a visual context attribute.

`QTVisualContextSetImageAvailableCallback` (page 129)
Installs a user-defined callback to receive notifications when a new image becomes available.

`QTVisualContextTask` (page 129)
Causes visual context to release internally held resources for later re-use.

Supporting Functions

`DisposeICMAlignmentUPP` (page 21)
Disposes of an ICMAlignmentUPP pointer.

`DisposeICMCompletionUPP` (page 22)
Disposes of an ICMCompletionUPP pointer.

`DisposeICMConvertDataFormatUPP` (page 22)
Disposes of an ICMConvertDataFormatUPP pointer.

`DisposeICMCursorShieldedUPP` (page 23)
Disposes of an ICMCursorShieldedUPP pointer.

`DisposeICMDataUPP` (page 23)
Disposes of an ICMDataUPP pointer.

`DisposeICMFlushUPP` (page 24)
Disposes of an ICMFlushUPP pointer.

`DisposeICMMemoryDisposedUPP` (page 24)
Disposes of an ICMMemoryDisposedUPP pointer.

`DisposeICMProgressUPP` (page 25)
  Disposes of an ICMProgressUPP pointer.

`DisposeQDPixUPP` (page 25)
  Disposes of a QDPixUPP pointer.

`DisposeStdPixUPP` (page 26)
  Disposes of a StdPixUPP pointer.

`ICMCompressionFrameOptionsCreate` (page 26)
  Creates a frame compression options object.

`ICMCompressionFrameOptionsCreateCopy` (page 27)
  Copies a frame compression options object.

`ICMCompressionFrameOptionsGetForceKeyFrame` (page 27)
  Retrieves the force key frame flag.

`ICMCompressionFrameOptionsGetFrameType` (page 28)
  Retrieves the frame type setting.

`ICMCompressionFrameOptionsGetProperty` (page 28)
  Retrieves the value of a specific property of a compression frame options object.

`ICMCompressionFrameOptionsGetPropertyInfo` (page 29)
  Retrieves information about properties of a compression frame options object.

`ICMCompressionFrameOptionsGetTypeID` (page 30)
  Returns the type ID for the current frame compression options object.

`ICMCompressionFrameOptionsRelease` (page 30)
  Decrements the retain count of a frame compression options object.

`ICMCompressionFrameOptionsRetain` (page 31)
  Increments the retain count of a frame compression options object.

`ICMCompressionFrameOptionsSetForceKeyFrame` (page 31)
  Forces frames to be compressed as key frames.

`ICMCompressionFrameOptionsSetFrameType` (page 32)
  Requests a frame be compressed as a particular frame type.

`ICMCompressionFrameOptionsSetProperty` (page 33)
  Sets the value of a specific property of a compression frame options object.

`ICMCompressionSessionBeginPass` (page 33)
  Announces the start of a specific compression pass.

`ICMCompressionSessionEndPass` (page 38)
  Announces the end of a pass.

`ICMCompressionSessionGetPropertyInfo` (page 41)
  Retrieves information about properties of a compression session.

`ICMCompressionSessionGetTimeScale` (page 42)
  Retrieves the time scale for a compression session.

`ICMCompressionSessionGetTypeID` (page 42)
  Returns the type ID for the current compression session.

`ICMCompressionSessionOptionsCreate` (page 42)
  Creates a compression session options object.

`ICMCompressionSessionOptionsCreateCopy` (page 43)
  Copies a compression session options object.

`ICMCompressionSessionOptionsGetAllowFrameReordering` (page 44)
: Retrieves the allow frame reordering flag.

`ICMCompressionSessionOptionsGetAllowFrameTimeChanges` (page 44)
: Retrieves the allow frame time changes flag.

`ICMCompressionSessionOptionsGetAllowTemporalCompression` (page 44)
: Retrieves the allow temporal compression flag.

`ICMCompressionSessionOptionsGetDurationsNeeded` (page 45)
: Retrieves the durations needed flag.

`ICMCompressionSessionOptionsGetMaxKeyFrameInterval` (page 45)
: Retrieves the maximum key frame interval.

`ICMCompressionSessionOptionsGetProperty` (page 46)
: Retrieves the value of a specific property of a compression session options object.

`ICMCompressionSessionOptionsGetPropertyInfo` (page 47)
: Retrieves information about properties of a compression session options object.

`ICMCompressionSessionOptionsGetTypeID` (page 48)
: Returns the type ID for the current compression session options object.

`ICMCompressionSessionOptionsRelease` (page 49)
: Decrements the retain count of a compression session options object.

`ICMCompressionSessionOptionsRetain` (page 49)
: Increments the retain count of a compression session options object.

`ICMCompressionSessionOptionsSetAllowFrameReordering` (page 49)
: Enables frame reordering.

`ICMCompressionSessionOptionsSetAllowFrameTimeChanges` (page 50)
: Allows the compressor to modify frame times.

`ICMCompressionSessionOptionsSetAllowTemporalCompression` (page 51)
: Enables temporal compression.

`ICMCompressionSessionOptionsSetDurationsNeeded` (page 51)
: Indicates that the durations of outputted frames must be calculated.

`ICMCompressionSessionOptionsSetMaxKeyFrameInterval` (page 52)
: Sets the maximum interval between key frames.

`ICMCompressionSessionOptionsSetProperty` (page 53)
: Sets the value of a specific property of a compression session options object.

`ICMCompressionSessionProcessBetweenPasses` (page 54)
: Lets the compressor perform processing between passes.

`ICMCompressionSessionRelease` (page 55)
: Decrements the retain count of a compression session.

`ICMCompressionSessionRetain` (page 55)
: Increments the retain count of a compression session.

`ICMCompressionSessionSetProperty` (page 56)
: Sets the value of a specific property of a compression session.

`ICMCompressionSessionSupportsMultiPassEncoding` (page 57)
: Queries whether a compression session supports multipass encoding.

`ICMCompressorSessionDropFrame` (page 57)
: Called by a compressor to notify the ICM that a source frame has been dropped and will not contribute to any encoded frames.

`ICMCompressorSessionEmitEncodedFrame` (page 58)
: Called by a compressor to output an encoded frame corresponding to one or more source frames.

`ICMCompressorSourceFrameGetDisplayNumber` (page 59)
: Retrieves a source frames display number.

`ICMCompressorSourceFrameGetDisplayTimeStampAndDuration` (page 59)
: Retrieves the display time stamp and duration of a source frame.

`ICMCompressorSourceFrameGetFrameOptions` (page 60)
: Retrieves the frame compression options for a source frame.

`ICMCompressorSourceFrameGetPixelBuffer` (page 60)
: Retrieves a source frames pixel buffer.

`ICMCompressorSourceFrameGetTypeID` (page 61)
: Returns the type ID for the current source frame object.

`ICMCompressorSourceFrameRelease` (page 61)
: Decrements the retain count of a source frame object.

`ICMCompressorSourceFrameRetain` (page 62)
: Increments the retain count of a source frame object.

`ICMDecompressionFrameOptionsCreate` (page 62)
: Creates a frame decompression options object.

`ICMDecompressionFrameOptionsCreateCopy` (page 63)
: Copies a frame decompression options object.

`ICMDecompressionFrameOptionsGetProperty` (page 63)
: Retrieves the value of a specific property of a decompression frame options object.

`ICMDecompressionFrameOptionsGetPropertyInfo` (page 64)
: Retrieves information about properties of a decompression frame options object.

`ICMDecompressionFrameOptionsGetTypeID` (page 65)
: Returns the type ID for the current frame decompression options object.

`ICMDecompressionFrameOptionsRelease` (page 66)
: Decrements the retain count of a frame decompression options object.

`ICMDecompressionFrameOptionsRetain` (page 66)
: Increments the retain count of a frame decompression options object.

`ICMDecompressionFrameOptionsSetProperty` (page 66)
: Sets the value of a specific property of a decompression frame options object.

`ICMDecompressionSessionCreate` (page 67)
: Creates a session for decompressing video frames.

`ICMDecompressionSessionCreateForVisualContext` (page 68)
: Creates a session for decompressing video frames.

`ICMDecompressionSessionDecodeFrame` (page 69)
: Queues a frame for decompression.

`ICMDecompressionSessionFlush` (page 70)
: Flushes the frames queued for a decompression session.

`ICMDecompressionSessionGetProperty` (page 71)
Retrieves the value of a specific property of a decompression session.

`ICMDecompressionSessionGetPropertyInfo` (page 72)
Retrieves information about the properties of a decompression session.

`ICMDecompressionSessionGetTypeID` (page 73)
Returns the type ID for the current decompression session.

`ICMDecompressionSessionOptionsCreate` (page 73)
Creates a decompression session options object.

`ICMDecompressionSessionOptionsCreateCopy` (page 74)
Copies a decompression session options object.

`ICMDecompressionSessionOptionsGetProperty` (page 74)
Retrieves the value of a specific property of a decompression session options object.

`ICMDecompressionSessionOptionsGetPropertyInfo` (page 75)
Retrieves information about properties of a decompression session options object.

`ICMDecompressionSessionOptionsGetTypeID` (page 76)
Returns the type ID for the current decompression session options object.

`ICMDecompressionSessionOptionsRelease` (page 76)
Decrements the retain count of a decompression session options object.

`ICMDecompressionSessionOptionsRetain` (page 77)
Increments the retain count of a decompression session options object.

`ICMDecompressionSessionOptionsSetProperty` (page 77)
Sets the value of a specific property of a decompression session options object.

`ICMDecompressionSessionRelease` (page 78)
Decrements the retain count of a decompression session.

`ICMDecompressionSessionRetain` (page 79)
Increments the retain count of a decompression session.

`ICMDecompressionSessionSetNonScheduledDisplayDirection` (page 79)
Sets the direction for non-scheduled display time.

`ICMDecompressionSessionSetNonScheduledDisplayTime` (page 80)
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.

`ICMDecompressionSessionSetProperty` (page 81)
Sets the value of a specific property of a decompression session.

`ICMEncodedFrameCreateMutable` (page 82)
Called by a compressor to create an encoded-frame token corresponding to a given source frame.

`ICMEncodedFrameGetBufferSize` (page 82)
Gets the size of an encoded frame's data buffer.

`ICMEncodedFrameGetDataPtr` (page 83)
Gets the data buffer for an encoded frame.

`ICMEncodedFrameGetDataSize` (page 83)
Gets the data size of the compressed frame in an encoded frame's buffer.

`ICMEncodedFrameGetDecodeDuration` (page 84)
Retrieves an encoded frame's decode duration.

`ICMEncodedFrameGetDecodeNumber` (page 84)
: Retrieves the decode number of an encoded frame.

`ICMEncodedFrameGetDecodeTimeStamp` (page 85)
: Retrieves an encoded frame's decode time stamp.

`ICMEncodedFrameGetDisplayDuration` (page 85)
: Retrieves an encoded frame's display duration.

`ICMEncodedFrameGetDisplayOffset` (page 85)
: Retrieves an encoded frame's display offset.

`ICMEncodedFrameGetDisplayTimeStamp` (page 86)
: Retrieves an encoded frame's display time stamp.

`ICMEncodedFrameGetFrameType` (page 86)
: Retrieves the frame type for an encoded frame.

`ICMEncodedFrameGetImageDescription` (page 87)
: Retrieves the image description of an encoded frame.

`ICMEncodedFrameGetMediaSampleFlags` (page 88)
: Retrieves the media sample flags for an encoded frame.

`ICMEncodedFrameGetSimilarity` (page 88)
: Retrieves the similarity value for an encoded frame.

`ICMEncodedFrameGetSourceFrameRefCon` (page 88)
: Retrieves the reference value of an encoded frame's source frame.

`ICMEncodedFrameGetTimeScale` (page 89)
: Retrieves the timescale of an encoded frame.

`ICMEncodedFrameGetTypeID` (page 89)
: Returns the type ID for the current encoded frame object.

`ICMEncodedFrameGetValidTimeFlags` (page 90)
: Retrieves an encoded frame's flags indicating which of its time stamps and durations are valid.

`ICMEncodedFrameRelease` (page 90)
: Decrements the retain count of an encoded frame object.

`ICMEncodedFrameRetain` (page 91)
: Increments the retain count of an encoded frame object.

`ICMEncodedFrameSetDataSize` (page 91)
: Sets the data size of the compressed frame in an encoded frame's buffer.

`ICMEncodedFrameSetDecodeDuration` (page 91)
: Sets an encoded frame's decode duration.

`ICMEncodedFrameSetDecodeTimeStamp` (page 92)
: Sets an encoded frame's decode time stamp.

`ICMEncodedFrameSetDisplayDuration` (page 92)
: Sets an encoded frame's display duration.

`ICMEncodedFrameSetDisplayTimeStamp` (page 93)
: Sets an encoded frame's display time stamp.

`ICMEncodedFrameSetFrameType` (page 93)
: Sets the frame type for an encoded frame.

`ICMEncodedFrameSetMediaSampleFlags` (page 94)
: Sets the media sample flags for an encoded frame.

`ICMEncodedFrameSetSimilarity` (page 95)
: Sets the similarity for an encoded frame.

`ICMEncodedFrameSetValidTimeFlags` (page 95)
: Sets an encoded frame's flags that indicate which of its time stamps and durations are valid.

`ICMImageDescriptionGetProperty` (page 96)
: Returns a particular property of a image description handle.

`ICMImageDescriptionGetPropertyInfo` (page 97)
: Returns information about a particular property of a image description.

`ICMImageDescriptionSetProperty` (page 97)
: Sets a particular property of a image description handle.

`ICMMultiPassStorageCopyDataAtTimeStamp` (page 98)
: Called by a multipass-capable compressor to retrieve data at a given time stamp.

`ICMMultiPassStorageCreateWithCallbacks` (page 99)
: Assembles a multipass storage mechanism from callbacks.

`ICMMultiPassStorageCreateWithTemporaryFile` (page 99)
: Creates multipass storage using a temporary file.

`ICMMultiPassStorageGetTimeStamp` (page 100)
: Called by a multipass-capable compressor to retrieve a time stamp for which a value is stored.

`ICMMultiPassStorageGetTypeID` (page 101)
: Returns the type ID for the current multipass storage object.

`ICMMultiPassStorageRelease` (page 101)
: Decrements the retain count of a multipass storage object.

`ICMMultiPassStorageRetain` (page 102)
: Increments the retain count of a multipass storage object.

`ICMMultiPassStorageSetDataAtTimeStamp` (page 102)
: Called by a multipass-capable compressor to store data at a given time stamp.

`NewICMAlignmentUPP` (page 106)
: Allocates a Universal Procedure Pointer for the ICMAlignmentProc callback.

`NewICMCompletionUPP` (page 107)
: Allocates a Universal Procedure Pointer for the ICMCompletionProc callback.

`NewICMConvertDataFormatUPP` (page 107)
: Allocates a Universal Procedure Pointer for the ICMConvertDataFormatProc callback.

`NewICMCursorShieldedUPP` (page 108)
: Allocates a Universal Procedure Pointer for the ICMCursorShieldedProc callback.

`NewICMDataUPP` (page 108)
: Allocates a Universal Procedure Pointer for the ICMDataProc callback.

`NewICMFlushUPP` (page 109)
: Allocates a Universal Procedure Pointer for the ICMFlushProc callback.

`NewICMMemoryDisposedUPP` (page 109)
: Allocates a Universal Procedure Pointer for the ICMMemoryDisposedProc callback.

`NewICMProgressUPP` (page 110)
: Allocates a Universal Procedure Pointer for the ICMProgressProc callback.

`NewQDPixUPP` (page 110)
: Allocates a Universal Procedure Pointer for the QDPixProc callback.

`NewStdPixUPP` (page 111)
: Allocates a Universal Procedure Pointer for the StdPixProc callback.

`QTAddComponentPropertyListener` (page 111)
: Installs a callback to monitor a component property.

`QTComponentPropertyListenerCollectionAddListener` (page 113)
: Adds a listener callback for a specified property class and ID to a property listener collection.

`QTComponentPropertyListenerCollectionCreate` (page 114)
: Creates a collection of component property monitors.

`QTComponentPropertyListenerCollectionHasListenersForProperty` (page 114)
: Determines if there are any listeners in a component property listener collection registered for a specified property class and ID.

`QTComponentPropertyListenerCollectionIsEmpty` (page 115)
: Determines if a listener collection is empty.

`QTComponentPropertyListenerCollectionNotifyListeners` (page 116)
: Calls all listener callbacks in a component property listener collection registered for a specified property class and ID.

`QTComponentPropertyListenerCollectionRemoveListener` (page 117)
: Removes a listener callback with a specified property class and ID from a property listener collection.

`QTGetComponentProperty` (page 118)
: Returns the value of a specific component property.

`QTGetComponentPropertyInfo` (page 120)
: Returns information about the properties of a component.

`QTPixelBufferContextCreate` (page 122)
: Creates a new pixel buffer context with the given attributes.

`QTRemoveComponentPropertyListener` (page 122)
: Removes a component property monitoring callback.

`QTSetComponentProperty` (page 123)
: Sets the value of a specific component property.

Functions

DisposeICMAlignmentUPP

Disposes of an ICMAlignmentUPP pointer.

```
void DisposeICMAlignmentUPP (
   ICMAlignmentUPP userUPP
);
```

Parameters

userUPP
: An `ICMAlignmentUPP` pointer. See `Universal Procedure Pointers`.

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

DisposeICMCompletionUPP

Disposes of an ICMCompletionUPP pointer.

```
void DisposeICMCompletionUPP (
   ICMCompletionUPP userUPP
);
```

Parameters

`userUPP`
    An `ICMCompletionUPP` pointer. See `Universal Procedure Pointers`.

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qtcompress

qtcompress.win

Declared In
`ImageCompression.h`

DisposeICMConvertDataFormatUPP

Disposes of an ICMConvertDataFormatUPP pointer.

```
void DisposeICMConvertDataFormatUPP (
   ICMConvertDataFormatUPP userUPP
);
```

Parameters

`userUPP`
    An `ICMConvertDataFormatUPP` pointer. See `Universal Procedure Pointers`.

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

DisposeICMCursorShieldedUPP

Disposes of an ICMCursorShieldedUPP pointer.

```
void DisposeICMCursorShieldedUPP (
   ICMCursorShieldedUPP userUPP
);
```

Parameters

`userUPP`

An `ICMCursorShieldedUPP` pointer. See `Universal Procedure Pointers`.

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

DisposeICMDataUPP

Disposes of an ICMDataUPP pointer.

```
void DisposeICMDataUPP (
   ICMDataUPP userUPP
);
```

Parameters

`userUPP`

An `ICMDataUPP` pointer. See `Universal Procedure Pointers`.

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

DisposeICMFlushUPP

Disposes of an ICMFlushUPP pointer.

```
void DisposeICMFlushUPP (
   ICMFlushUPP userUPP
);
```

Parameters

`userUPP`
An `ICMFlushUPP` pointer. See `Universal Procedure Pointers`.

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

DisposeICMMemoryDisposedUPP

Disposes of an ICMMemoryDisposedUPP pointer.

```
void DisposeICMMemoryDisposedUPP (
   ICMMemoryDisposedUPP userUPP
);
```

Parameters

`userUPP`
An `ICMMemoryDisposedUPP` pointer. See `Universal Procedure Pointers`.

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

DisposeICMProgressUPP

Disposes of an ICMProgressUPP pointer.

```
void DisposeICMProgressUPP (
   ICMProgressUPP userUPP
);
```

Parameters

userUPP

An `ICMProgressUPP` pointer. See `Universal Procedure Pointers`.

Return Value

You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes

Introduced in QuickTime 4.1.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qtdataexchange

qtdataexchange.win

Declared In

`ImageCompression.h`

DisposeQDPixUPP

Disposes of a QDPixUPP pointer.

```
void DisposeQDPixUPP (
   QDPixUPP userUPP
);
```

Parameters

userUPP

A `QDPixUPP` pointer. See `Universal Procedure Pointers`.

Return Value

You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes

Introduced in QuickTime 4.1.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

DisposeStdPixUPP

Disposes of a StdPixUPP pointer.

```
void DisposeStdPixUPP (
   StdPixUPP userUPP
);
```

Parameters

userUPP
: A `StdPixUPP` pointer. See `Universal Procedure Pointers`.

Return Value

You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes

Introduced in QuickTime 4.1.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Desktop Sprites

DesktopSprites

DesktopSprites.win

Declared In

`ImageCompression.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 35).

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

Return Value

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

Availability

Available in Mac OS X v10.3 and later.

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`.

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.

Availability
Available in Mac OS X v10.3 and later.

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`.

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

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
ExampleIPBCodec

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`.

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

Discussion
This function can return one of these constants:

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
ExampleIPBCodec

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`.

`inPropClass`

Pass the following constant to define the property class: `kComponentPropertyClassPropertyInfo` = `'pnfo'` The property information class. See these constants:

`kComponentPropertyClassPropertyInfo`

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: `kComponentPropertyCacheFlagNotPersistentProperty` metadata should not be saved in persistent cache. `kComponentPropertyCacheFlagIsDynamicProperty` metadata should not cached at all. `kComponentPropertyExtendedInfo` = `'meta'` A `CFDictionary` with extended property information. See these constants:

`kComponentPropertyInfoList`

`kComponentPropertyCacheSeed`

`kComponentPropertyCacheFlags`

`kComponentPropertyExtendedInfo`

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.

Availability

Available in Mac OS X v10.3 and later.

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`.

inPropClass

Pass the following constant to define the property class: `kComponentPropertyClassPropertyInfo` = `'pnfo'` The property information class. See these constants:

`kComponentPropertyClassPropertyInfo`

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: `kComponentPropertyCacheFlagNotPersistentProperty` metadata should not be saved in persistent cache. `kComponentPropertyCacheFlagIsDynamicProperty` metadata should not cached at all. `kComponentPropertyExtendedInfo` = `'meta'` A `CFDictionary` with extended property information. See these constants:

- `kComponentPropertyInfoList`
- `kComponentPropertyCacheSeed`
- `kComponentPropertyCacheFlags`
- `kComponentPropertyExtendedInfo`

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.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`ImageCompression.h`

ICMCompressionFrameOptionsGetTypeID

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

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

Return Value
A `CFTypeID` value.

Availability
Available in Mac OS X v10.3 and later.

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`. If you pass NULL, nothing happens.

Discussion

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

Availability

Available in Mac OS X v10.3 and later.

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`. If you pass NULL, nothing happens.

Return Value

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

Availability

Available in Mac OS X v10.3 and later.

Declared In

`ImageCompression.h`

ICMCompressionFrameOptionsSetForceKeyFrame

Forces frames to be compressed as key frames.

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

Parameters

options

A compression frame options reference. This reference is returned by `ICMCompressionFrameOptionsCreate`.

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.

Availability
Available in Mac OS X v10.3 and later.

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`.

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. See these constants:

`kICMFrameType_I`

`kICMFrameType_P`

`kICMFrameType_B`

`kICMFrameType_Unknown`

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`.

Availability
Available in Mac OS X v10.3 and later.

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`.

inPropClass

Pass the following constant to define the property class: `kComponentPropertyClassPropertyInfo` = `'pnfo'` The property information class. See these constants:

`kComponentPropertyClassPropertyInfo`

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: `kComponentPropertyCacheFlagNotPersistentProperty` metadata should not be saved in persistent cache. `kComponentPropertyCacheFlagIsDynamicProperty` metadata should not cached at all. `kComponentPropertyExtendedInfo` = `'meta'` A `CFDictionary` with extended property information. See these constants:

`kComponentPropertyInfoList`

`kComponentPropertyCacheSeed`

`kComponentPropertyCacheFlags`

`kComponentPropertyExtendedInfo`

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.

Availability

Available in Mac OS X v10.3 and later.

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 35).

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. See these constants:

`kICMCompressionPassMode_OutputEncodedFrames`

`kICMCompressionPassMode_NoSourceFrames`

`kICMCompressionPassMode_WriteToMultiPassStorage`

`kICMCompressionPassMode_ReadFromMultiPassStorage`

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` (page 37), because they are used to index the compressor's stored state.

Availability

Available in Mac OS X v10.3 and later.

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 35).

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` (page 37).

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` (page 37), pass in `nextDisplayTimeStamp` the display timestamp of the next frame that would be passed to `EncodeFrame`.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

CaptureAndCompressIPBMovie

OpenGLCaptureToMovie

Quartz Composer QCTV

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.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
CaptureAndCompressIPBMovie
OpenGLCaptureToMovie
Quartz Composer QCTV

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 35).

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` (page 35). 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` (page 35). 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. See these constants:

`kICMValidTime_DisplayTimeStampIsValid`

`kICMValidTime_DisplayDurationIsValid`

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`.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
CaptureAndCompressIPBMovie
OpenGLCaptureToMovie
Quartz Composer QCTV

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 35).

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

Availability
Available in Mac OS X v10.3 and later.

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 35).

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.

Availability

Available in Mac OS X v10.3 and later.

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 35).

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 35).

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` (page 35). 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.

Availability

Available in Mac OS X v10.3 and later.

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 35).

inPropClass

Pass the following constant to define the property class: `kComponentPropertyClassPropertyInfo` = `'pnfo'` The property information class. See these constants:

`kComponentPropertyClassPropertyInfo`

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: `kComponentPropertyCacheFlagNotPersistentProperty` metadata should not be saved in persistent cache. `kComponentPropertyCacheFlagIsDynamicProperty` metadata should not cached at all. `kComponentPropertyExtendedInfo` = `'meta'` A `CFDictionary` with extended property information. See these constants:

`kComponentPropertyInfoList`

`kComponentPropertyCacheSeed`

`kComponentPropertyCacheFlags`

`kComponentPropertyExtendedInfo`

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.

Availability

Available in Mac OS X v10.3 and later.

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 35).

inPropClass

Pass the following constant to define the property class: `kComponentPropertyClassPropertyInfo` = `'pnfo'` The property information class. See these constants:

`kComponentPropertyClassPropertyInfo`

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: `kComponentPropertyCacheFlagNotPersistentProperty` metadata should not be saved in persistent cache. `kComponentPropertyCacheFlagIsDynamicProperty` metadata should not cached at all. `kComponentPropertyExtendedInfo` = `'meta'` A `CFDictionary` with extended property information. See these constants:

`kComponentPropertyInfoList`

`kComponentPropertyCacheSeed`

`kComponentPropertyCacheFlags`

`kComponentPropertyExtendedInfo`

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.

Availability

Available in Mac OS X v10.3 and later.

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 35).

Return Value

The time scale for the compression session.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

OpenGLCaptureToMovie

Quartz Composer QCTV

Declared In

`ImageCompression.h`

ICMCompressionSessionGetTypeID

Returns the type ID for the current compression session.

```
CFTypeID ICMCompressionSessionGetTypeID (
   void
);
```

Return Value

A `CFTypeID` value.

Availability

Available in Mac OS X v10.3 and later.

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.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
CaptureAndCompressIPBMovie
Quartz Composer QCTV

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`.

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.

Availability
Available in Mac OS X v10.3 and later.

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`.

Return Value

Returns TRUE if frame reordering is allowed, FALSE otherwise.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

ExampleIPBCodec

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`.

Return Value

Returns TRUE if the compressor is allowed to modify frame times, FALSE otherwise.

Availability

Available in Mac OS X v10.3 and later.

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`.

Return Value

Returns TRUE if temporal compression is allowed, FALSE otherwise.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

ExampleIPBCodec

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`.

Return Value

Returns TRUE if the durations of outputted frames must be calculated, FALSE otherwise.

Availability

Available in Mac OS X v10.3 and later.

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`.

Return Value

Returns the maximum key frame interval.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

ExampleIPBCodec

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`.

inPropClass

Pass the following constant to define the property class: `kComponentPropertyClassPropertyInfo` = `'pnfo'` The property information class. See these constants:

`kComponentPropertyClassPropertyInfo`

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: `kComponentPropertyCacheFlagNotPersistentProperty` metadata should not be saved in persistent cache. `kComponentPropertyCacheFlagIsDynamicProperty` metadata should not cached at all. `kComponentPropertyExtendedInfo` = `'meta'` A `CFDictionary` with extended property information. See these constants:

- `kComponentPropertyInfoList`
- `kComponentPropertyCacheSeed`
- `kComponentPropertyCacheFlags`
- `kComponentPropertyExtendedInfo`

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.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
ExampleIPBCodec

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`.

inPropClass
Pass the following constant to define the property class: `kComponentPropertyClassPropertyInfo` = `'pnfo'` The property information class. See these constants:
`kComponentPropertyClassPropertyInfo`

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: `kComponentPropertyCacheFlagNotPersistentProperty` metadata should not be saved in persistent cache. `kComponentPropertyCacheFlagIsDynamicProperty` metadata should not cached at all. `kComponentPropertyExtendedInfo` = `'meta'` A `CFDictionary` with extended property information. See these constants:
`kComponentPropertyInfoList`
`kComponentPropertyCacheSeed`
`kComponentPropertyCacheFlags`
`kComponentPropertyExtendedInfo`

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.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`ImageCompression.h`

ICMCompressionSessionOptionsGetTypeID

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

```
CFTypeID ICMCompressionSessionOptionsGetTypeID (
   void
);
```

Return Value
A `CFTypeID` value.

Availability
Available in Mac OS X v10.3 and later.

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`. If you pass NULL, nothing happens.

Discussion

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

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

CaptureAndCompressIPBMovie

ExampleIPBCodec

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`. If you pass NULL, nothing happens.

Return Value

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

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

ExampleIPBCodec

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`.

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.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

CaptureAndCompressIPBMovie

OpenGLCaptureToMovie

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`.

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.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
CaptureAndCompressIPBMovie
OpenGLCaptureToMovie

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`.

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.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
CaptureAndCompressIPBMovie
OpenGLCaptureToMovie

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`.

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.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

CaptureAndCompressIPBMovie

OpenGLCaptureToMovie

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`.

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.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
CaptureAndCompressIPBMovie

OpenGLCaptureToMovie

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`.

inPropClass

Pass the following constant to define the property class: `kComponentPropertyClassPropertyInfo` = `'pnfo'` The property information class. See these constants:

`kComponentPropertyClassPropertyInfo`

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: `kComponentPropertyCacheFlagNotPersistentProperty` metadata should not be saved in persistent cache. `kComponentPropertyCacheFlagIsDynamicProperty` metadata should not cached at all. `kComponentPropertyExtendedInfo` = `'meta'` A `CFDictionary` with extended property information. See these constants:

`kComponentPropertyInfoList`

`kComponentPropertyCacheSeed`

`kComponentPropertyCacheFlags`

`kComponentPropertyExtendedInfo`

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.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
CaptureAndCompressIPBMovie
OpenGLCaptureToMovie
Quartz Composer QCTV

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 35).

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. See these constants:

- `kICMCompressionPassMode_OutputEncodedFrames`
- `kICMCompressionPassMode_NoSourceFrames`
- `kICMCompressionPassMode_WriteToMultiPassStorage`
- `kICMCompressionPassMode_ReadFromMultiPassStorage`

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.

Availability
Available in Mac OS X v10.3 and later.

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 35). If you pass NULL, nothing happens.

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

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
CaptureAndCompressIPBMovie
OpenGLCaptureToMovie
Quartz Composer QCTV

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 35). If you pass NULL, nothing happens.

Return Value
A reference to the object passed in session, for convenience.

Availability
Available in Mac OS X v10.3 and later.

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 35).

inPropClass
Pass the following constant to define the property class: `kComponentPropertyClassPropertyInfo` = `'pnfo'` The property information class. See these constants:

`kComponentPropertyClassPropertyInfo`

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: `kComponentPropertyCacheFlagNotPersistentProperty` metadata should not be saved in persistent cache. `kComponentPropertyCacheFlagIsDynamicProperty` metadata should not cached at all. `kComponentPropertyExtendedInfo` = `'meta'` A `CFDictionary` with extended property information. See these constants:

`kComponentPropertyInfoList`

`kComponentPropertyCacheSeed`

`kComponentPropertyCacheFlags`

`kComponentPropertyExtendedInfo`

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.

Availability
Available in Mac OS X v10.3 and later.

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 35).

`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`.

Availability
Available in Mac OS X v10.3 and later.

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 37). 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`.

Availability

Available in Mac OS X v10.3 and later.

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

session

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 37).

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`.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

ExampleIPBCodec

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 37).

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 (page 37). 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.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
ExampleIPBCodec

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 37).

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.

See these constants:

`kICMValidTime_DisplayTimeStampIsValid`

`kICMValidTime_DisplayDurationIsValid`

Return Value

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

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

ExampleIPBCodec

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 37).

Return Value

A compression session frame options reference representing options for this frame. A frame options object is created by `ICMCompressionFrameOptionsCreate`.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

ExampleIPBCodec

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 37).

Return Value

A reference to the pixel buffer containing the source frame's image being compressed.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

ExampleIPBCodec

Declared In

`ImageCompression.h`

ICMCompressorSourceFrameGetTypeID

Returns the type ID for the current source frame object.

```
CFTypeID ICMCompressorSourceFrameGetTypeID (
   void
);
```

Return Value

A `CFTypeID` value.

Availability

Available in Mac OS X v10.3 and later.

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 37). If you pass NULL, nothing happens.

Discussion

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

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
ExampleIPBCodec

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 37). If you pass NULL, nothing happens.

Return Value
A reference to the object passed in `sourceFrame`, for convenience.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
ExampleIPBCodec

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.

Availability
Available in Mac OS X v10.3 and later.

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`.

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.

Availability
Available in Mac OS X v10.3 and later.

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`.

inPropClass

Pass the following constant to define the property class: `kComponentPropertyClassPropertyInfo` = `'pnfo'` The property information class. See these constants:

`kComponentPropertyClassPropertyInfo`

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: `kComponentPropertyCacheFlagNotPersistentProperty` metadata should not be saved in persistent cache. `kComponentPropertyCacheFlagIsDynamicProperty` metadata should not cached at all. `kComponentPropertyExtendedInfo` = `'meta'` A `CFDictionary` with extended property information. See these constants:

`kComponentPropertyInfoList`

`kComponentPropertyCacheSeed`

`kComponentPropertyCacheFlags`

`kComponentPropertyExtendedInfo`

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.

Availability

Available in Mac OS X v10.3 and later.

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`.

inPropClass

Pass the following constant to define the property class: `kComponentPropertyClassPropertyInfo` = `'pnfo'` The property information class. See these constants:

`kComponentPropertyClassPropertyInfo`

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: `kComponentPropertyCacheFlagNotPersistentProperty` metadata should not be saved in persistent cache. `kComponentPropertyCacheFlagIsDynamicProperty` metadata should not cached at all. `kComponentPropertyExtendedInfo` = `'meta'` A `CFDictionary` with extended property information. See these constants:

`kComponentPropertyInfoList`

`kComponentPropertyCacheSeed`

`kComponentPropertyCacheFlags`

`kComponentPropertyExtendedInfo`

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.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`ImageCompression.h`

ICMDecompressionFrameOptionsGetTypeID

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

```
CFTypeID ICMDecompressionFrameOptionsGetTypeID (
   void
);
```

Return Value

A `CFTypeID` value.

Availability

Available in Mac OS X v10.3 and later.

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`. If you pass NULL, nothing happens.

Discussion

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

Availability

Available in Mac OS X v10.3 and later.

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`. If you pass NULL, nothing happens.

Return Value

A reference to the frame decompression options object passed in options, for convenience.

Availability

Available in Mac OS X v10.3 and later.

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`.

inPropClass

Pass the following constant to define the property class: `kComponentPropertyClassPropertyInfo` = `'pnfo'` The property information class. See these constants:

`kComponentPropertyClassPropertyInfo`

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: `kComponentPropertyCacheFlagNotPersistentProperty` metadata should not be saved in persistent cache. `kComponentPropertyCacheFlagIsDynamicProperty` metadata should not cached at all. `kComponentPropertyExtendedInfo` = `'meta'` A `CFDictionary` with extended property information. See these constants:

`kComponentPropertyInfoList`

`kComponentPropertyCacheSeed`

`kComponentPropertyCacheFlags`

`kComponentPropertyExtendedInfo`

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.

Availability

Available in Mac OS X v10.3 and later.

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`. 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` and `ICMDecompressionTrackingCallbackProc`.

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`.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
CaptureAndCompressIPBMovie

MovieVideoChart

WhackedTV

Declared In
`ImageCompression.h`

ICMDecompressionSessionCreateForVisualContext

Creates a session for decompressing video frames.

```
OSStatus ICMDecompressionSessionCreateForVisualContext (
   CFAllocatorRef allocator,
   ImageDescriptionHandle desc,
   ICMDecompressionSessionOptionsRef decompressionOptions,
   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.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

QTQuartzPlayer

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`.

data

A pointer to the compressed data for this frame. The data must remain in this location until `ICMDecompressionTrackingCallbackProc` 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`.

frameTime

A pointer to a structure describing the frame's timing information.

sourceFrameRefCon

Your reference value for the frame.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

CaptureAndCompressIPBMovie

MovieVideoChart

QTQuartzPlayer

WhackedTV

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`.

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.

Availability
Available in Mac OS X v10.3 and later.

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`.

`inPropClass`

Pass the following constant to define the property class: `kComponentPropertyClassPropertyInfo` = `'pnfo'` The property information class. See these constants:

`kComponentPropertyClassPropertyInfo`

`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: `kComponentPropertyCacheFlagNotPersistentProperty` metadata should not be saved in persistent cache. `kComponentPropertyCacheFlagIsDynamicProperty` metadata should not cached at all. `kComponentPropertyExtendedInfo` = `'meta'` A `CFDictionary` with extended property information. See these constants:

`kComponentPropertyInfoList`

`kComponentPropertyCacheSeed`

`kComponentPropertyCacheFlags`

`kComponentPropertyExtendedInfo`

`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.

Availability
Available in Mac OS X v10.3 and later.

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`.

`inPropClass`
Pass the following constant to define the property class: `kComponentPropertyClassPropertyInfo` = `'pnfo'` The property information class. See these constants:
`kComponentPropertyClassPropertyInfo`

`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: `kComponentPropertyCacheFlagNotPersistentProperty` metadata should not be saved in persistent cache. `kComponentPropertyCacheFlagIsDynamicProperty` metadata should not cached at all. `kComponentPropertyExtendedInfo` = `'meta'` A `CFDictionary` with extended property information. See these constants:
`kComponentPropertyInfoList`
`kComponentPropertyCacheSeed`
`kComponentPropertyCacheFlags`
`kComponentPropertyExtendedInfo`

`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.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`ImageCompression.h`

ICMDecompressionSessionGetTypeID

Returns the type ID for the current decompression session.

```
CFTypeID ICMDecompressionSessionGetTypeID (
   void
);
```

Return Value
A `CFTypeID` value.

Availability
Available in Mac OS X v10.3 and later.

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.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
WhackedTV

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`.

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.

Availability

Available in Mac OS X v10.3 and later.

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`.

inPropClass

Pass the following constant to define the property class: `kComponentPropertyClassPropertyInfo` = `'pnfo'` The property information class. See these constants:

`kComponentPropertyClassPropertyInfo`

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: `kComponentPropertyCacheFlagNotPersistentProperty` metadata should not be saved in persistent cache. `kComponentPropertyCacheFlagIsDynamicProperty` metadata should not cached at all. `kComponentPropertyExtendedInfo` = `'meta'` A `CFDictionary` with extended property information. See these constants:

`kComponentPropertyInfoList`

`kComponentPropertyCacheSeed`

`kComponentPropertyCacheFlags`

`kComponentPropertyExtendedInfo`

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.

Availability

Available in Mac OS X v10.3 and later.

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`.

inPropClass

Pass the following constant to define the property class: `kComponentPropertyClassPropertyInfo` = `'pnfo'` The property information class. See these constants:

`kComponentPropertyClassPropertyInfo`

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: `kComponentPropertyCacheFlagNotPersistentProperty` metadata should not be saved in persistent cache. `kComponentPropertyCacheFlagIsDynamicProperty` metadata should not cached at all. `kComponentPropertyExtendedInfo` = `'meta'` A `CFDictionary` with extended property information. See these constants:

- `kComponentPropertyInfoList`
- `kComponentPropertyCacheSeed`
- `kComponentPropertyCacheFlags`
- `kComponentPropertyExtendedInfo`

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.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`ImageCompression.h`

ICMDecompressionSessionOptionsGetTypeID

Returns the type ID for the current decompression session options object.

```
CFTypeID ICMDecompressionSessionOptionsGetTypeID (
   void
);
```

Return Value
A `CFTypeID` value.

Availability
Available in Mac OS X v10.3 and later.

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`. If you pass NULL, nothing happens.

Discussion

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

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

CaptureAndCompressIPBMovie

WhackedTV

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`. If you pass NULL, nothing happens.

Return Value

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

Availability

Available in Mac OS X v10.3 and later.

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`.

inPropClass

Pass the following constant to define the property class: `kComponentPropertyClassPropertyInfo` = `'pnfo'` The property information class. See these constants:

`kComponentPropertyClassPropertyInfo`

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: `kComponentPropertyCacheFlagNotPersistentProperty` metadata should not be saved in persistent cache. `kComponentPropertyCacheFlagIsDynamicProperty` metadata should not cached at all. `kComponentPropertyExtendedInfo` = `'meta'` A `CFDictionary` with extended property information. See these constants:

`kComponentPropertyInfoList`

`kComponentPropertyCacheSeed`

`kComponentPropertyCacheFlags`

`kComponentPropertyExtendedInfo`

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.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

WhackedTV

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`. If you pass NULL, nothing happens.

Discussion

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

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

CaptureAndCompressIPBMovie

MovieVideoChart

QTQuartzPlayer

WhackedTV

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`. If you pass NULL, nothing happens.

Return Value

A copy of the reference passed in session, for convenience.

Availability

Available in Mac OS X v10.3 and later.

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`.

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.

Availability

Available in Mac OS X v10.3 and later.

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`.

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.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

CaptureAndCompressIPBMovie

QTQuartzPlayer

WhackedTV

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`.

`inPropClass`

Pass the following constant to define the property class: `kComponentPropertyClassPropertyInfo` = `'pnfo'` The property information class. See these constants:

`kComponentPropertyClassPropertyInfo`

`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: `kComponentPropertyCacheFlagNotPersistentProperty` metadata should not be saved in persistent cache. `kComponentPropertyCacheFlagIsDynamicProperty` metadata should not cached at all. `kComponentPropertyExtendedInfo` = `'meta'` A `CFDictionary` with extended property information. See these constants:

`kComponentPropertyInfoList`

`kComponentPropertyCacheSeed`

`kComponentPropertyCacheFlags`

`kComponentPropertyExtendedInfo`

`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.

Availability
Available in Mac OS X v10.3 and later.

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 37).

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`.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
ExampleIPBCodec

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.

Availability
Available in Mac OS X v10.3 and later.

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.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
CaptureAndCompressIPBMovie
ExampleIPBCodec
Quartz Composer QCTV

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.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
CaptureAndCompressIPBMovie
Quartz Composer QCTV

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.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
CaptureAndCompressIPBMovie
OpenGLCaptureToMovie
Quartz Composer QCTV

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.

Availability
Available in Mac OS X v10.3 and later.

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.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
CaptureAndCompressIPBMovie

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.

Availability
Available in Mac OS X v10.3 and later.

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.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

CaptureAndCompressIPBMovie

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.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

CaptureAndCompressIPBMovie

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:

Availability

Available in Mac OS X v10.3 and later.

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` (page 38).

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

CaptureAndCompressIPBMovie

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`.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

CaptureAndCompressIPBMovie

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.

Availability

Available in Mac OS X v10.3 and later.

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 37).

Availability

Available in Mac OS X v10.3 and later.

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.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

CaptureAndCompressIPBMovie

Declared In

`ImageCompression.h`

ICMEncodedFrameGetTypeID

Returns the type ID for the current encoded frame object.

```
CFTypeID ICMEncodedFrameGetTypeID (
   void
);
```

Return Value

A `CFTypeID` value.

Availability

Available in Mac OS X v10.3 and later.

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:

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
CaptureAndCompressIPBMovie

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.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
ExampleIPBCodec

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.

Availability

Available in Mac OS X v10.3 and later.

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.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

ExampleIPBCodec

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.

Availability

Available in Mac OS X v10.3 and later.

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.

Availability

Available in Mac OS X v10.3 and later.

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.

Availability

Available in Mac OS X v10.3 and later.

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.

Availability

Available in Mac OS X v10.3 and later.

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. See these constants:

`kICMFrameType_I`

`kICMFrameType_P`

`kICMFrameType_B`

`kICMFrameType_Unknown`

Return Value

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

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

ExampleIPBCodec

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.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

ExampleIPBCodec

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.

Availability
Available in Mac OS X v10.3 and later.

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. See these constants:

`kICMValidTime_DisplayTimeStampIsValid`

`kICMValidTime_DisplayDurationIsValid`

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.

Availability
Available in Mac OS X v10.3 and later.

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.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
CaptureAndCompressIPBMovie

WhackedTV

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,
   ByteCount *outPropValueSize,
   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.

Availability

Available in Mac OS X v10.3 and later.

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.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

ExampleIPBCodec

SoftVDigX

WhackedTV

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.

Availability

Available in Mac OS X v10.3 and later.

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`.

multiPassStorageOut

A reference to the new multipass storage object.

Return Value

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

Availability

Available in Mac OS X v10.3 and later.

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. See these constants:

`kICMMultiPassStorage_DoNotDeleteWhenDone`

multiPassStorageOut

A reference to the new multipass storage.

Return Value

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

Availability

Available in Mac OS X v10.3 and later.

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. See these constants:

`kICMMultiPassStorage_GetFirstTimeStamp`

`kICMMultiPassStorage_GetPreviousTimeStamp`

`kICMMultiPassStorage_GetNextTimeStamp`

`kICMMultiPassStorage_GetLastTimeStamp`

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.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`ImageCompression.h`

ICMMultiPassStorageGetTypeID

Returns the type ID for the current multipass storage object.

```
CFTypeID ICMMultiPassStorageGetTypeID (
   void
);
```

Return Value
A `CFTypeID` value.

Availability
Available in Mac OS X v10.3 and later.

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` or `ICMMultiPassStorageCreateWithCallbacks`. If you pass NULL, nothing happens.

Discussion

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

Availability

Available in Mac OS X v10.3 and later.

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` or `ICMMultiPassStorageCreateWithCallbacks`. If you pass NULL, nothing happens.

Return Value

A reference to the object passed in `multiPassStorage`, for convenience.

Availability

Available in Mac OS X v10.3 and later.

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.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`ImageCompression.h`

ImageTranscoderBeginSequence

Initiates an image transcoding sequence and specifies the input data format.

```
ComponentResult ImageTranscoderBeginSequence (
   ImageTranscoderComponent itc,
   ImageDescriptionHandle srcDesc,
   ImageDescriptionHandle *dstDesc,
   void *data,
   long dataSize
);
```

Parameters

itc
: The image transcoder component.

srcDesc
: The `ImageDescription` structure for the source compressed image data.

dstDesc
: On return, a new `ImageDescription` structure.

data
: First frame of data to be transcoded (may be `NIL`).

dataSize
: `Size` of compressed image data pointed to by the data.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function specifies the format of source compressed image data in the `srcDesc` parameter. The image transcoder should allocate a new `ImageDescription` structure and return it in the `dstDesc` parameter. The new `ImageDescription` structure should be a completely filled out image description which is sufficient for correctly decompressing the data generated by subsequent calls to `ImageTranscoderConvert` (page 104).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

ImageTranscoderConvert

Performs image transcoding operations.

```
ComponentResult ImageTranscoderConvert (
   ImageTranscoderComponent itc,
   void *srcData,
   long srcDataSize,
   void **dstData,
   long *dstDataSize
);
```

Parameters

itc
: The image transcoder component.

srcData
: A pointer to the source compressed image data to transcode.

srcDataSize
: The size of the source image data, in bytes.

dstData
: On return, a pointer to the transcoded data.

dstDataSize
: On return, the size of the transcoded data in bytes.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The image transcoder component is responsible for allocating storage for the transcoded data, transcoding the data, and returning a pointer to the transcoded data in the `dstData` parameter. The size of the transcoded data in bytes should be returned in the `dstDataSize` parameter. The caller is responsible for disposing of the transcoded data using `ImageTranscoderDisposeData` (page 105).

The memory allocated to store the transcoded image data must not be in an unlocked handle. Even if the image transcoding operation can be performed in place, the transcoded data must be placed in a separate block of memory from the source data. The image transcoder component must not write back into the source image data.

The responsibility for allocating the buffer for the transcoded data has been placed in the transcoder with the intent that some hardware manufacturers may find it useful to place the transcoded data directly into on-board memory on their video board. If the transcoding operation is being performed on a QuickTime movie, the transcoded data pointer will be almost immediately passed on to a decompressor. If the decompressor is implemented in hardware, performance may be increased because the transcoded data is already loaded onto the decompression hardware.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

ImageTranscoderDisposeData

Disposes of transcoded data.

```
ComponentResult ImageTranscoderDisposeData (
   ImageTranscoderComponent itc,
   void *dstData
);
```

Parameters

itc

The image transcoder component.

dstData

A pointer to the transcoded data.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
When the client of the image transcoder component is done with a piece of transcoded data, this function must be called with a pointer to the transcoded data. The image transcoder component should not make any assumptions about the maximum number of outstanding pieces of transcoded data or the order in which the transcoding data will be disposed.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

ImageTranscoderEndSequence

Ends an image transcoding sequence.

```
ComponentResult ImageTranscoderEndSequence (
   ImageTranscoderComponent itc
);
```

Parameters

itc

The image transcoder component whose transcoder sequence is ending.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

`ImageTranscoderEndSequence` is called when there are no more frames of data to be transcoded using the parameters specified in the previous call to `ImageTranscoderBeginSequence` (page 103). After calling this function, the component will either be closed or receive another call to `ImageTranscoderBeginSequence` with a different `ImageDescription` structure. For example, the dimensions of the source image may be different.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

NewICMAlignmentUPP

Allocates a Universal Procedure Pointer for the ICMAlignmentProc callback.

```
ICMAlignmentUPP NewICMAlignmentUPP (
   ICMAlignmentProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your application-defined function.

Return Value

A new UPP; see `Universal Procedure Pointers`.

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes

Introduced in QuickTime 4.1. Replaces `NewICMAlignmentProc`.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

NewICMCompletionUPP

Allocates a Universal Procedure Pointer for the ICMCompletionProc callback.

```
ICMCompletionUPP NewICMCompletionUPP (
   ICMCompletionProcPtr userRoutine
);
```

Parameters

`userRoutine`
A pointer to your application-defined function.

Return Value
A new UPP; see `Universal Procedure Pointers`.

Discussion
This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes
Introduced in QuickTime 4.1. Replaces `NewICMCompletionProc`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

NewICMConvertDataFormatUPP

Allocates a Universal Procedure Pointer for the ICMConvertDataFormatProc callback.

```
ICMConvertDataFormatUPP NewICMConvertDataFormatUPP (
   ICMConvertDataFormatProcPtr userRoutine
);
```

Parameters

`userRoutine`
A pointer to your application-defined function.

Return Value
A new UPP; see `Universal Procedure Pointers`.

Discussion
This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes
Introduced in QuickTime 4.1. Replaces `NewICMConvertDataFormatProc`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

NewICMCursorShieldedUPP

Allocates a Universal Procedure Pointer for the ICMCursorShieldedProc callback.

```
ICMCursorShieldedUPP NewICMCursorShieldedUPP (
   ICMCursorShieldedProcPtr userRoutine
);
```

Parameters

`userRoutine`
A pointer to your application-defined function.

Return Value
A new UPP; see `Universal Procedure Pointers`.

Discussion
This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes
Introduced in QuickTime 4.1. Replaces `NewICMCursorShieldedProc`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

NewICMDataUPP

Allocates a Universal Procedure Pointer for the ICMDataProc callback.

```
ICMDataUPP NewICMDataUPP (
   ICMDataProcPtr userRoutine
);
```

Parameters

`userRoutine`
A pointer to your application-defined function.

Return Value
A new UPP; see `Universal Procedure Pointers`.

Discussion
This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes
Introduced in QuickTime 4.1. Replaces `NewICMDataProc`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

NewICMFlushUPP

Allocates a Universal Procedure Pointer for the ICMFlushProc callback.

```
ICMFlushUPP NewICMFlushUPP (
   ICMFlushProcPtr userRoutine
);
```

Parameters

`userRoutine`

A pointer to your application-defined function.

Return Value
A new UPP; see `Universal Procedure Pointers`.

Discussion
This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes
Introduced in QuickTime 4.1. Replaces `NewICMFlushProc`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

NewICMMemoryDisposedUPP

Allocates a Universal Procedure Pointer for the ICMMemoryDisposedProc callback.

```
ICMMemoryDisposedUPP NewICMMemoryDisposedUPP (
   ICMMemoryDisposedProcPtr userRoutine
);
```

Parameters

`userRoutine`

A pointer to your application-defined function.

Return Value
A new UPP; see `Universal Procedure Pointers`.

Discussion
This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes
Introduced in QuickTime 4.1. Replaces `NewICMMemoryDisposedProc`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

NewICMProgressUPP

Allocates a Universal Procedure Pointer for the ICMProgressProc callback.

```
ICMProgressUPP NewICMProgressUPP (
   ICMProgressProcPtr userRoutine
);
```

Parameters

userRoutine
A pointer to your application-defined function.

Return Value
A new UPP; see `Universal Procedure Pointers`.

Discussion
This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes
Introduced in QuickTime 4.1. Replaces `NewICMProgressProc`.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qtdataexchange

qtdataexchange.win

ThreadsExporter

ThreadsImporter

Declared In
`ImageCompression.h`

NewQDPixUPP

Allocates a Universal Procedure Pointer for the QDPixProc callback.

```
QDPixUPP NewQDPixUPP (
   QDPixProcPtr userRoutine
);
```

Parameters

userRoutine
A pointer to your application-defined function.

Return Value
A new UPP; see `Universal Procedure Pointers`.

Discussion
This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes
Introduced in QuickTime 4.1. Replaces `NewQDPixProc`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

NewStdPixUPP

Allocates a Universal Procedure Pointer for the StdPixProc callback.

```
StdPixUPP NewStdPixUPP (
   StdPixProcPtr userRoutine
);
```

Parameters

`userRoutine`
A pointer to your application-defined function.

Return Value
A new UPP; see `Universal Procedure Pointers`.

Discussion
This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes
Introduced in QuickTime 4.1. Replaces `NewStdPixProc`.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Desktop Sprites

DesktopSprites

DesktopSprites.win

Declared In
`ImageCompression.h`

QTAddComponentPropertyListener

Installs a callback to monitor a component property.

```
ComponentResult QTAddComponentPropertyListener (
   ComponentInstance inComponent,
   ComponentPropertyClass inPropClass,
   ComponentPropertyID inPropID,
   QTComponentPropertyListenerUPP inDispatchProc,
   void *inUserData
);
```

Parameters

inComponent

A component instance, which you can get by calling `OpenComponent` or `OpenDefaultComponent`.

inPropClass

A value (see below) of type `OSType` that specifies a property class: `kComponentPropertyClassPropertyInfo` (`'pnfo'`) A `QTComponentPropertyInfo` structure that defines a property information class. `kComponentPropertyInfoList` (`'list'`) An array of `QTComponentPropertyInfo` structures, one for each property. `kComponentPropertyCacheSeed` (`'seed'`) A component property cache seed value. `kComponentPropertyExtendedInfo` (`'meta'`) A `CFDictionary` with extended property information. `kComponentPropertyCacheFlags` (`'flgs'`) One of the following two flags: `kComponentPropertyCacheFlagNotPersistent` Property metadata should not be saved in persistent cache. `kComponentPropertyCacheFlagIsDynamic` Property metadata should not be cached at all. See these constants:

`kComponentPropertyClassPropertyInfo`

`kComponentPropertyInfoList`

`kComponentPropertyCacheSeed`

`kComponentPropertyExtendedInfo`

`kComponentPropertyCacheFlags`

`kComponentPropertyCacheFlagNotPersistent`

`kComponentPropertyCacheFlagIsDynamic`

inPropID

A value of type `OSType` that specifies a property ID.

inDispatchProc

A Universal Procedure Pointer to a `QTComponentPropertyListenerProc` callback.

inUserData

A pointer to user data that will be passed to the callback. You may pass NULL in this parameter.

Return Value

See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 6.4.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

WhackedTV

Declared In

`ImageCompression.h`

QTComponentPropertyListenerCollectionAddListener

Adds a listener callback for a specified property class and ID to a property listener collection.

```
OSStatus QTComponentPropertyListenerCollectionAddListener (
   QTComponentPropertyListenersRef inCollection,
   ComponentPropertyClass inPropClass,
   ComponentPropertyID inPropID,
   QTComponentPropertyListenerUPP inListenerProc,
   const void *inListenerProcRefCon
);
```

Parameters

inCollection

A property listener collection created by a previous call to `QTComponentPropertyListenerCollectionCreate`.

inPropClass

A value (see below) of type `OSType` that specifies a property class: `kComponentPropertyClassPropertyInfo` (`'pnfo'`) A `QTComponentPropertyInfo` structure that defines a property information class. `kComponentPropertyInfoList` (`'list'`) An array of `QTComponentPropertyInfo` structures, one for each property. `kComponentPropertyCacheSeed` (`'seed'`) A component property cache seed value. `kComponentPropertyExtendedInfo` (`'meta'`) A `CFDictionary` with extended property information. `kComponentPropertyCacheFlags` (`'flgs'`) One of the following two flags: `kComponentPropertyCacheFlagNotPersistent` Property metadata should not be saved in persistent cache. `kComponentPropertyCacheFlagIsDynamic` Property metadata should not be cached at all. See these constants:

`kComponentPropertyClassPropertyInfo`

`kComponentPropertyInfoList`

`kComponentPropertyCacheSeed`

`kComponentPropertyExtendedInfo`

`kComponentPropertyCacheFlags`

`kComponentPropertyCacheFlagNotPersistent`

`kComponentPropertyCacheFlagIsDynamic`

inPropID

A value of type `OSType` that specifies a property ID.

inListenerProc

A `QTComponentPropertyListenerProc` callback.

inListenerProcRefCon

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

Return Value

See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 6.4.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`ImageCompression.h`

QTComponentPropertyListenerCollectionCreate

Creates a collection of component property monitors.

```
OSStatus QTComponentPropertyListenerCollectionCreate (
   CFAllocatorRef inAllocator,
   const QTComponentPropertyListenerCollectionContext *inContext,
   QTComponentPropertyListenersRef *outCollection
);
```

Parameters

inAllocator

A pointer to the allocator used to create the collection and its contents. You can pass `NIL`.

inContext

A pointer to a `QTComponentPropertyInfo` data structure. You can pass `NIL` if no structure exists. A copy of the contents of the structure is made; therefore you can pass a pointer to a structure on the stack.

outCollection

On return, a pointer to the new empty listener collection.

Return Value

See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 6.4.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`ImageCompression.h`

QTComponentPropertyListenerCollectionHasListenersForProperty

Determines if there are any listeners in a component property listener collection registered for a specified property class and ID.

```
Boolean QTComponentPropertyListenerCollectionHasListenersForProperty (
   QTComponentPropertyListenersRef inCollection,
   ComponentPropertyClass inPropClass,
   ComponentPropertyID inPropID
);
```

Parameters

inCollection

A property listener collection created by a previous call to `QTComponentPropertyListenerCollectionCreate`.

inPropClass

A value (see below) of type `OSType` that specifies a property class: `kComponentPropertyClassPropertyInfo` (`'pnfo'`) A `QTComponentPropertyInfo` structure that defines a property information class. `kComponentPropertyInfoList` (`'list'`) An array of `QTComponentPropertyInfo` structures, one for each property. `kComponentPropertyCacheSeed` (`'seed'`) A component property cache seed value. `kComponentPropertyExtendedInfo` (`'meta'`) A `CFDictionary` with extended property information. `kComponentPropertyCacheFlags` (`'flgs'`) One of the following two flags: `kComponentPropertyCacheFlagNotPersistent` Property metadata should not be saved in persistent cache. `kComponentPropertyCacheFlagIsDynamic` Property metadata should not be cached at all. See these constants:

`kComponentPropertyClassPropertyInfo`

`kComponentPropertyInfoList`

`kComponentPropertyCacheSeed`

`kComponentPropertyExtendedInfo`

`kComponentPropertyCacheFlags`

`kComponentPropertyCacheFlagNotPersistent`

`kComponentPropertyCacheFlagIsDynamic`

inPropID

A value of type `OSType` that specifies a property ID.

Return Value
Returns TRUE if there are any listeners in the listener collection registered for the specified property class and ID, FALSE otherwise.

Version Notes
Introduced in QuickTime 6.4.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`ImageCompression.h`

QTComponentPropertyListenerCollectionIsEmpty

Determines if a listener collection is empty.

```
Boolean QTComponentPropertyListenerCollectionIsEmpty (
   QTComponentPropertyListenersRef inCollection
);
```

Parameters

inCollection

A property listener collection created by a previous call to `QTComponentPropertyListenerCollectionCreate`.

Return Value
Returns TRUE if the collection is empty, FALSE otherwise.

Version Notes
Introduced in QuickTime 6.4.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`ImageCompression.h`

QTComponentPropertyListenerCollectionNotifyListeners

Calls all listener callbacks in a component property listener collection registered for a specified property class and ID.

```
OSStatus QTComponentPropertyListenerCollectionNotifyListeners (
   QTComponentPropertyListenersRef inCollection,
   ComponentInstance inNotifier,
   ComponentPropertyClass inPropClass,
   ComponentPropertyID inPropID,
   const void *inFilterProcRefCon,
   UInt32 inFlags
);
```

Parameters

inCollection
A property listener collection created by a previous call to `QTComponentPropertyListenerCollectionCreate`.

inNotifier
The caller's component instance.

inPropClass
A value (see below) of type `OSType` that specifies a property class: `kComponentPropertyClassPropertyInfo` (`'pnfo'`) A `QTComponentPropertyInfo` structure that defines a property information class. `kComponentPropertyInfoList` (`'list'`) An array of `QTComponentPropertyInfo` structures, one for each property. `kComponentPropertyCacheSeed` (`'seed'`) A component property cache seed value. `kComponentPropertyExtendedInfo` (`'meta'`) A `CFDictionary` with extended property information. `kComponentPropertyCacheFlags` (`'flgs'`) One of the following two flags: `kComponentPropertyCacheFlagNotPersistent` Property metadata should not be saved in persistent cache. `kComponentPropertyCacheFlagIsDynamic` Property metadata should not be cached at all. See these constants:

`kComponentPropertyClassPropertyInfo`
`kComponentPropertyInfoList`
`kComponentPropertyCacheSeed`
`kComponentPropertyExtendedInfo`
`kComponentPropertyCacheFlags`
`kComponentPropertyCacheFlagNotPersistent`
`kComponentPropertyCacheFlagIsDynamic`

inPropID
A value of type `OSType` that specifies a property ID.

inFilterProcRefCon
A reference constant to be passed to your callback. Use this parameter to point to a data structure containing any information your function needs. You may pass `NIL`.

inFlags

Currently not used.

Return Value

See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error.

Discussion

If the `filterProcUPP` field in the `QTComponentPropertyListenerCollectionContext` data structure that was passed to `QTComponentPropertyListenerCollectionCreate` is not `NIL`, the `QTComponentPropertyListenerFilterProc` callback it points to will be called before each call to a registered listener that matches the specified property class and ID passed to this function. If the filter function return FALSE, that listener callback will not be called. This lets a component change the calling semantics (for example, to call another thread) or use a different listener callback signature.

Version Notes

Introduced in QuickTime 6.4.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`ImageCompression.h`

QTComponentPropertyListenerCollectionRemoveListener

Removes a listener callback with a specified property class and ID from a property listener collection.

```
OSStatus QTComponentPropertyListenerCollectionRemoveListener (
   QTComponentPropertyListenersRef inCollection,
   ComponentPropertyClass inPropClass,
   ComponentPropertyID inPropID,
   QTComponentPropertyListenerUPP inListenerProc,
   const void *inListenerProcRefCon
);
```

Parameters

inCollection

A property listener collection created by a previous call to `QTComponentPropertyListenerCollectionCreate`.

inPropClass

A value (see below) of type `OSType` that specifies a property class: `kComponentPropertyClassPropertyInfo` (`'pnfo'`) A `QTComponentPropertyInfo` structure that defines a property information class. `kComponentPropertyInfoList` (`'list'`) An array of `QTComponentPropertyInfo` structures, one for each property. `kComponentPropertyCacheSeed` (`'seed'`) A component property cache seed value. `kComponentPropertyExtendedInfo` (`'meta'`) A `CFDictionary` with extended property information. `kComponentPropertyCacheFlags` (`'flgs'`) One of the following two flags: `kComponentPropertyCacheFlagNotPersistent` Property metadata should not be saved in persistent cache. `kComponentPropertyCacheFlagIsDynamic` Property metadata should not be cached at all. See these constants:

- `kComponentPropertyClassPropertyInfo`
- `kComponentPropertyInfoList`
- `kComponentPropertyCacheSeed`
- `kComponentPropertyExtendedInfo`
- `kComponentPropertyCacheFlags`
- `kComponentPropertyCacheFlagNotPersistent`
- `kComponentPropertyCacheFlagIsDynamic`

inPropID

A value of type `OSType` that specifies a property ID.

inListenerProc

The `QTComponentPropertyListenerProc` callback to be removed.

inListenerProcRefCon

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

Return Value

See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 6.4.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`ImageCompression.h`

QTGetComponentProperty

Returns the value of a specific component property.

```
ComponentResult QTGetComponentProperty (
   ComponentInstance inComponent,
   ComponentPropertyClass inPropClass,
   ComponentPropertyID inPropID,
   ByteCount inPropValueSize,
   ComponentValuePtr outPropValueAddress,
   ByteCount *outPropValueSizeUsed
);
```

Parameters

inComponent

A component instance, which you can get by calling `OpenComponent` or `OpenDefaultComponent`.

inPropClass

A value (see below) of type `OSType` that specifies a property class: `kComponentPropertyClassPropertyInfo` (`'pnfo'`) A `QTComponentPropertyInfo` structure that defines a property information class. `kComponentPropertyInfoList` (`'list'`) An array of `QTComponentPropertyInfo` structures, one for each property. `kComponentPropertyCacheSeed` (`'seed'`) A component property cache seed value. `kComponentPropertyExtendedInfo` (`'meta'`) A `CFDictionary` with extended property information. `kComponentPropertyCacheFlags` (`'flgs'`) One of the following two flags: `kComponentPropertyCacheFlagNotPersistent` Property metadata should not be saved in persistent cache. `kComponentPropertyCacheFlagIsDynamic` Property metadata should not be cached at all. See these constants:

`kComponentPropertyClassPropertyInfo`

`kComponentPropertyInfoList`

`kComponentPropertyCacheSeed`

`kComponentPropertyExtendedInfo`

`kComponentPropertyCacheFlags`

`kComponentPropertyCacheFlagNotPersistent`

`kComponentPropertyCacheFlagIsDynamic`

inPropID

A value of type `OSType` that specifies 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

See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 6.4.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

CaptureAndCompressIPBMovie

QTExtractAndConvertToAIFF

QTExtractAndConvertToMovieFile

SCAudioCompress
WhackedTV

Declared In
`ImageCompression.h`

QTGetComponentPropertyInfo

Returns information about the properties of a component.

```
ComponentResult QTGetComponentPropertyInfo (
   ComponentInstance inComponent,
   ComponentPropertyClass inPropClass,
   ComponentPropertyID inPropID,
   ComponentValueType *outPropType,
   ByteCount *outPropValueSize,
   UInt32 *outPropertyFlags
);
```

Parameters

inComponent

A component instance, which you can get by calling `OpenComponent` or `OpenDefaultComponent`.

inPropClass

A value (see below) of type `OSType` that specifies a property class: `kComponentPropertyClassPropertyInfo` (`'pnfo'`) A `QTComponentPropertyInfo` structure that defines a property information class. `kComponentPropertyInfoList` (`'list'`) An array of `QTComponentPropertyInfo` structures, one for each property. `kComponentPropertyCacheSeed` (`'seed'`) A component property cache seed value. `kComponentPropertyExtendedInfo` (`'meta'`) A `CFDictionary` with extended property information. `kComponentPropertyCacheFlags` (`'flgs'`) One of the following two flags: `kComponentPropertyCacheFlagNotPersistent` Property metadata should not be saved in persistent cache. `kComponentPropertyCacheFlagIsDynamic` Property metadata should not be cached at all. See these constants:

`kComponentPropertyClassPropertyInfo`

`kComponentPropertyInfoList`

`kComponentPropertyCacheSeed`

`kComponentPropertyExtendedInfo`

`kComponentPropertyCacheFlags`

`kComponentPropertyCacheFlagNotPersistent`

`kComponentPropertyCacheFlagIsDynamic`

inPropID

A value of type `OSType` that specifies a property ID.

outPropType

A pointer to memory allocated to hold the `property` type on return. This pointer may be NULL.

outPropValueSize

A pointer to memory allocated to hold the size of the property value on return. This pointer may be NULL.

outPropertyFlags

A pointer to memory allocated to hold property flags on return.

Return Value
See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 6.4.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
ElectricImageComponent
QTExtractAndConvertToAIFF
QTExtractAndConvertToMovieFile
SCAudioCompress
WhackedTV

Declared In
`ImageCompression.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,
   QTVisualContextRef *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.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
LiveVideoMixer2

LiveVideoMixer3

QTCoreImage101

QTCoreVideo103

QTCoreVideo201

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.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
QTCoreVideo102

QTCoreVideo103

QTCoreVideo201

QTCoreVideo301

QTPixelBufferVCToCGImage

Declared In
`ImageCompression.h`

QTRemoveComponentPropertyListener

Removes a component property monitoring callback.

```
ComponentResult QTRemoveComponentPropertyListener (
   ComponentInstance inComponent,
   ComponentPropertyClass inPropClass,
   ComponentPropertyID inPropID,
   QTComponentPropertyListenerUPP inDispatchProc,
   void *inUserData
);
```

Parameters

inComponent

A component instance, which you can get by calling `OpenComponent` or `OpenDefaultComponent`.

inPropClass

A value (see below) of type `OSType` that specifies a property class: `kComponentPropertyClassPropertyInfo` (`'pnfo'`) A `QTComponentPropertyInfo` structure that defines a property information class. `kComponentPropertyInfoList` (`'list'`) An array of `QTComponentPropertyInfo` structures, one for each property. `kComponentPropertyCacheSeed` (`'seed'`) A component property cache seed value. `kComponentPropertyExtendedInfo` (`'meta'`) A `CFDictionary` with extended property information. `kComponentPropertyCacheFlags` (`'flgs'`) One of the following two flags: `kComponentPropertyCacheFlagNotPersistent` Property metadata should not be saved in persistent cache. `kComponentPropertyCacheFlagIsDynamic` Property metadata should not be cached at all. See these constants:

- `kComponentPropertyClassPropertyInfo`
- `kComponentPropertyInfoList`
- `kComponentPropertyCacheSeed`
- `kComponentPropertyExtendedInfo`
- `kComponentPropertyCacheFlags`
- `kComponentPropertyCacheFlagNotPersistent`
- `kComponentPropertyCacheFlagIsDynamic`

inPropID

A value of type `OSType` that specifies a property ID.

inDispatchProc

A Universal Procedure Pointer to a `QTComponentPropertyListenerProc` callback.

inUserData

User data to be passed to the callback.

Return Value

See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 6.4.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`ImageCompression.h`

QTSetComponentProperty

Sets the value of a specific component property.

```
ComponentResult QTSetComponentProperty (
   ComponentInstance inComponent,
   ComponentPropertyClass inPropClass,
   ComponentPropertyID inPropID,
   ByteCount inPropValueSize,
   ConstComponentValuePtr inPropValueAddress
);
```

Parameters

inComponent

A component instance, which you can get by calling `OpenComponent` or `OpenDefaultComponent`.

inPropClass

A value of type `OSType` that specifies a property class: `kComponentPropertyClassPropertyInfo` (`'pnfo'`) A `QTComponentPropertyInfo` structure that defines a property information class. `kComponentPropertyInfoList` (`'list'`) An array of `QTComponentPropertyInfo` structures, one for each property. `kComponentPropertyCacheSeed` (`'seed'`) A component property cache seed value. `kComponentPropertyExtendedInfo` (`'meta'`) A `CFDictionary` with extended property information. `kComponentPropertyCacheFlags` (`'flgs'`) One of the following two flags: `kComponentPropertyCacheFlagNotPersistent` Property metadata should not be saved in persistent cache. `kComponentPropertyCacheFlagIsDynamic` Property metadata should not be cached at all. See these constants:

- `kComponentPropertyClassPropertyInfo`
- `kComponentPropertyInfoList`
- `kComponentPropertyCacheSeed`
- `kComponentPropertyExtendedInfo`
- `kComponentPropertyCacheFlags`
- `kComponentPropertyCacheFlagNotPersistent`
- `kComponentPropertyCacheFlagIsDynamic`

inPropID

A value of type `OSType` that specifies 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.

Return Value

See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 6.4.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

CaptureAndCompressIPBMovie

QTExtractAndConvertToAIFF

QTExtractAndConvertToMovieFile

SCAudioCompress

WhackedTV

Declared In
`ImageCompression.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`.

`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.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
QTCoreImage101
QTCoreVideo103
QTCoreVideo201
QTPixelBufferVCToCGImage
VideoViewer

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.

Availability
Available in Mac OS X v10.3 and later.

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`.

Availability
Available in Mac OS X v10.3 and later.

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.

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` (page 125).

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

LiveVideoMixer2

QTCoreImage101

QTCoreVideo103

QTCoreVideo201

QTPixelBufferVCToCGImage

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.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

LiveVideoMixer3

QTCoreVideo103

QTCoreVideo201

QTPixelBufferVCToCGImage

QTQuartzPlayer

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.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
QTQuartzPlayer

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.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
CIVideoDemoGL
VideoViewer

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.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
QTPixelBufferVCToCGImage

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.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

LiveVideoMixer2

QTCoreImage101

QTCoreVideo103

QTCoreVideo201

QTPixelBufferVCToCGImage

Declared In

`ImageCompression.h`

Callbacks

ICMAlignmentProc

Provides an alignment behavior for windows based on the screen's bit depth.

```
typedef void (*ICMAlignmentProcPtr) (Rect *rp, long refcon);
```

If you name your function `MyICMAlignmentProc`, you would declare it this way:

```
void MyICMAlignmentProc (
    Rect    *rp,
    long    refcon );
```

Parameters

rp

Contains a pointer to a rectangle that has already been aligned with a default alignment function.

refcon

Contains a reference constant value for use by your alignment function. Your application specifies the value of this reference constant in the alignment function structure you pass to the Image Compression Manager.

Declared In

`ImageCompression.h`

ICMCompletionProc

Called by a compressor component upon completion of an asynchronous operation.

```
typedef void (*ICMCompletionProcPtr) (OSErr result, short flags, long refcon);
```

If you name your function `MyICMCompletionProc`, you would declare it this way:

```
void MyICMCompletionProc (
    OSErr    result,
    short    flags,
    long     refcon );
```

Parameters

result

Indicator of success of current operation.

flags

Contains flags (see below) that indicate which part of the operation is complete. Note that more than one of the flags may be set to 1. See these constants:

`codecCompletionSource`

`codecCompletionDest`

refcon

Contains a reference constant value for use by your completion function. Your application specifies the value of this reference constant in the callback function structure you pass to the Image Compression Manager.

Declared In

`ImageCompression.h`

ICMCursorShieldedProc

Undocumented

```
typedef void (*ICMCursorShieldedProcPtr) (const Rect *r, void *refcon, long flags);
```

If you name your function `MyICMCursorShieldedProc`, you would declare it this way:

```
void MyICMCursorShieldedProc (
    const Rect    *r,
    void          *refcon,
    long          flags );
```

Parameters

r

Undocumented

refcon

Pointer to a reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

flags

Undocumented

Declared In

`ImageCompression.h`

ICMDataProc

Supplies compressed data during a decompression operation.

```
typedef OSErr (*ICMDataProcPtr) (Ptr *dataP, long bytesNeeded, long refcon);
```

If you name your function `MyICMDataProc`, you would declare it this way:

```
OSErr MyICMDataProc (
    Ptr      *dataP,
    long     bytesNeeded,
    long     refcon );
```

Parameters

dataP

Contains a pointer to the address of the data buffer. The decompressor uses this parameter to indicate where your data-loading function should return the compressed data. You establish this data buffer when you start the decompression operation. For example, the `data` parameter to `FDecompressImage` (page 643) defines the location of the data buffer for that operation. Upon return from your data-loading function, this pointer should refer to the beginning of the compressed data that you loaded. The decompressor may also use this parameter to indicate that it wants to reset the mark within the compressed data stream. If the `dataP` parameter is set to `NIL`, the `bytesNeeded` parameter contains the new mark position, relative to the current position of the data stream. If your data-loading function does not support this operation, return a nonzero result code.

bytesNeeded

Specifies the number of bytes requested or the new mark offset. If the decompressor has requested additional compressed data (that is, the value of the `dataP` parameter is not `NIL`), then this parameter specifies how many bytes to return. This value never exceeds the size of the original data buffer. Your data-loading function should read the data from the current mark in the input data stream. If the decompressor has requested to set a new mark position in the data stream (that is, the value of the `dataP` parameter is `NIL`), then this parameter specifies the new mark position relative to the current position of the data stream.

refcon

Contains a reference constant value for use by your data-loading function. Your application specifies the value of this reference constant in the data-loading function structure you pass to the Image Compression Manager.

Return Value

See `Error Codes`. Your callback should return `noErr` if there is no error.

Declared In

`ImageCompression.h`

ICMFlushProc

Writes compressed data to a storage device during a compression operation.

```
typedef OSErr (*ICMFlushProcPtr) (Ptr data, long bytesAdded, long refcon);
```

If you name your function `MyICMFlushProc`, you would declare it this way:

```
OSErr MyICMFlushProc (
    Ptr     data,
    long    bytesAdded,
    long    refcon );
```

Parameters

data

Points to the data buffer. The compressor uses this parameter to indicate where your data-unloading function can find the compressed data. You establish this data buffer when you start the compression operation. For example, the `data` parameter to `FCompressImage` (page 637) defines the location of the data buffer for that operation. This pointer contains a 32-bit clean address. Your `ICMFlushProc` function should make no other assumptions about the value of this address. The compressor may also use this parameter to indicate that it wants to reset the mark within the compressed data stream. If the `data` parameter is set to `NIL`, the `bytesNeeded` parameter contains the new mark position, relative to the current position of the output data stream. If your `ICMFlushProc` function does not support this operation, return a nonzero result code.

bytesAdded

Specifies the number of bytes to write or the new mark offset. If the compressor wants to write out some compressed data (that is, the value of data is not `NIL`), then this parameter specifies how many bytes to write. This value never exceeds the size of the original data buffer. Your `ICMFlushProc` function should write that data at the current mark in the output data stream. If the compressor has requested to set a new mark position in the output data stream (that is, the value of data is `NIL`), then this parameter specifies the new mark position relative to the current position of the data stream.

refcon

Contains a reference constant value for use by your `ICMFlushProc` function. Your application specifies the value of this reference constant in the data-unloading function structure you pass to the Image Compression Manager.

Return Value

See `Error Codes`. Your callback should return `noErr` if there is no error.

Discussion

You assign an `ICMFlushProc` function to an image or a sequence by passing a pointer to a structure that identifies the function to the appropriate compression function.

Declared In

`ImageCompression.h`

ICMProgressProc

Reports on the progress of a compressor or decompressor.

```
typedef OSErr (*ICMProgressProcPtr) (short message, Fixed completeness, long refcon);
```

If you name your function `MyICMProgressProc`, you would declare it this way:

```
OSErr MyICMProgressProc (
    short    message,
    Fixed    completeness,
    long      refcon );
```

Parameters

message

Indicates why the Image Compression Manager called your function. There are three valid messages, listed below. See these constants:

`codecProgressOpen`

`codecProgressUpdatePercent`

`codecProgressClose`

completeness

Contains a fixed-point value indicating how far the operation has progressed. Its value is always between 0.0 and 1.0. This parameter is valid only when the `message` field is set to `codecProgressUpdatePercent`.

refcon

Contains a reference constant value for use by your progress function. Your application specifies the value of this reference constant in the progress function structure you pass to the Image Compression Manager.

Return Value

See `Error Codes`. Your callback should return `noErr` if there is no error. When a component calls your progress function, it supplies you with a number that indicates the completion percentage. Your program can cause the component to terminate the current operation by returning a result code of `codecAbortErr`.

Discussion

The Image Compression Manager calls your progress function only during long operations, and it does not call your function more than 30 times per second.

Declared In

`ImageCompression.h`

QDPixProc

Undocumented

```
typedef void (*QDPixProcPtr) (PixMap *src, Rect *srcRect, MatrixRecord *matrix,
short mode, RgnHandle mask, PixMap *matte, Rect *matteRect,
short flags);
```

If you name your function `MyQDPixProc`, you would declare it this way:

```
void MyQDPixProc (
    PixMap          *src,
    Rect            *srcRect,
    MatrixRecord    *matrix,
    short           mode,
    RgnHandle       mask,
    PixMap          *matte,
    Rect            *matteRect,
    short           flags );
```

Parameters

src

Undocumented

srcRect

Undocumented

matrix

Undocumented

mode

Undocumented

mask

Undocumented

matte

Undocumented

matteRect

Undocumented

flags

Undocumented

Declared In

`ImageCompression.h`

StdPixProc

Undocumented

```
typedef void (*StdPixProcPtr) (PixMap *src, Rect *srcRect, MatrixRecord *matrix,
short mode, RgnHandle mask, PixMap *matte, Rect *matteRect,
short flags);
```

If you name your function `MyStdPixProc`, you would declare it this way:

```
void MyStdPixProc (
    PixMap          *src,
    Rect            *srcRect,
    MatrixRecord    *matrix,
    short           mode,
    RgnHandle       mask,
    PixMap          *matte,
    Rect            *matteRect,
    short           flags );
```

Parameters

src

Undocumented

srcRect

Undocumented

matrix

Undocumented

`mode`
: *Undocumented*

`mask`
: *Undocumented*

`matte`
: *Undocumented*

`matteRect`
: *Undocumented*

`flags`
: *Undocumented*

Declared In
`ImageCompression.h`

Data Types

ICMAlignmentUPP

Represents a type used by the Image Compression API.

```
typedef STACK_UPP_TYPE(ICMAlignmentProcPtr) ICMAlignmentUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

ICMCompletionUPP

Represents a type used by the Image Compression API.

```
typedef STACK_UPP_TYPE(ICMCompletionProcPtr) ICMCompletionUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

ICMCursorShieldedUPP

Represents a type used by the Image Compression API.

```
typedef STACK_UPP_TYPE(ICMCursorShieldedProcPtr) ICMCursorShieldedUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

ICMDataUPP

Represents a type used by the Image Compression API.

```
typedef STACK_UPP_TYPE(ICMDataProcPtr) ICMDataUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

ICMDecompressionTrackingCallbackRecord

Designates a tracking callback for an ICM decompression session.

```
struct ICMDecompressionTrackingCallbackRecord {
ICMDecompressionTrackingCallback    decompressionTrackingCallback;
void                                *decompressionTrackingRefCon;
};
```

Fields
`decompressionTrackingCallback`
Discussion
The callback function pointer. See `ICMDecompressionTrackingCallbackProc`.

`decompressionTrackingRefCon`
Discussion
The callback's reference value.

Declared In
`ImageCompression.h`

ICMFlushUPP

Represents a type used by the Image Compression API.

```
typedef STACK_UPP_TYPE(ICMFlushProcPtr) ICMFlushUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`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`

Discussion
The version of this structure. Set to `kICMMultiPassStorageCallbacksVersionOne`.

`storageRefCon`

Discussion
A pointer to a reference constant. Use this parameter to point to a data structure containing any information your callback needs.

`setDataAtTimeStampCallback`

Discussion
A callback for storing values.

`getTimeStampCallback`

Discussion
A callback for finding time stamps.

`copyDataAtTimeStampCallback`

Discussion
A callback for retrieving values.

`releaseCallback`

Discussion
A callback for disposing the callback's state when done.

Discussion
This structure is used by `ICMMultiPassStorageCreateWithCallbacks`.

Declared In
`ImageCompression.h`

ICMProgressUPP

Represents a type used by the Image Compression API.

```
typedef STACK_UPP_TYPE(ICMProgressProcPtr) ICMProgressUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

ImageTranscoderComponent

Represents a type used by the Image Compression API.

```
typedef ComponentInstance ImageTranscoderComponent;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

QDPixUPP

Represents a type used by the Image Compression API.

```
typedef STACK_UPP_TYPE(QDPixProcPtr) QDPixUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

QTComponentPropertyListenerCollectionContext

Provides context information for a QTComponentPropertyListenerFilterProc callback.

```
struct QTComponentPropertyListenerCollectionContext
{
    UInt32                                  version;
    QTComponentPropertyListenerFilterUPP    filterProcUPP;
    void                                    *filterProcData;
};
```

Fields

`version`

Discussion
The version of this callback.

`filterProcUPP`

Discussion
A Universal Procedure Pointer to a `QTComponentPropertyListenerFilterProc` callback.

`filterProcData`

Discussion
A pointer to data for the callback.

Version Notes
Introduced in QuickTime 6.4.

Related Functions
Associated function:
`QTComponentPropertyListenerCollectionNotifyListeners` (page 116)

Declared In
`ImageCompression.h`

StdPixUPP

Represents a type used by the Image Compression API.

```
typedef STACK_UPP_TYPE(StdPixProcPtr) StdPixUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

Constants

ICMProgressProc Values

Constants passed to ICMProgressProc.

```
enum {
  codecProgressOpen              = 0,
  codecProgressUpdatePercent     = 1,
  codecProgressClose             = 2
};
```

Constants

`codecProgressOpen`

Indicates the start of a long operation. This is always the first message sent to your function. Your function can use this message to trigger the display of your progress window.

Available in Mac OS X v10.0 and later.

Declared in `ImageCompression.h`.

`codecProgressUpdatePercent`

Passes completion information to your function. The Image Compression Manager repeatedly sends this message to your function. The completeness parameter indicates the relative completion of the operation. You can use this value to update your progress window.

Available in Mac OS X v10.0 and later.

Declared in `ImageCompression.h`.

Declared In
`ImageCompression.h`

ICM Property IDs

Constants that contain the IDs of ICM properties.

```
enum {
  /*
   * Both fields should be decompressed.
   */
  kICMFieldMode_BothFields      = 0,
  /*
   * Only the top field should be decompressed, producing a half-height
   * image.
   */
  kICMFieldMode_TopFieldOnly    = 1,
  /*
   * Only the bottom field should be decompressed, producing a
   * half-height image.
   */
  kICMFieldMode_BottomFieldOnly = 2,
  /*
   * Both fields should be decompressed, and then filtered to reduce
   * interlacing artifacts.
   */
  kICMFieldMode_DeinterlaceFields = 3
};
enum {
  /*
   * Class identifier for compression frame options object properties.
   */
  kQTPropertyClass_ICMCompressionFrameOptions = 'icfo',
  /*
   * Forces frames to be compressed as key frames.
   * The compressor must obey the "force key frame" flag if set. By
   * default this property is false.
   */
  kICMCompressionFrameOptionsPropertyID_ForceKeyFrame = 'keyf', /* Boolean,
Read/Write */
  /*
   * Requests a frame be compressed as a particular frame type.
   *  The frame type setting may be ignored by the compressor if not
   * appropriate.
   * By default this is set to kICMFrameType_Unknown.
   * Do not assume that kICMFrameType_I means a key frame; if you need
   * a key frame, set the "force key frame" property.
   */
  kICMCompressionFrameOptionsPropertyID_FrameType = 'frty' /* ICMFrameType,
Read/Write */
};
enum {
  /*
   * Class identifier for compression session options object properties.
   */
  kQTPropertyClass_ICMCompressionSessionOptions = 'icso',
  /*
   * Enables temporal compression. By default, temporal compression is
   * disabled.
   * IMPORTANT: If you want temporal compression (P frames and/or B
   * frames) you must set this to true.
   */
  kICMCompressionSessionOptionsPropertyID_AllowTemporalCompression = 'p ok', /*
Boolean, Read/Write */
  /*
```

```
   * Enables frame reordering.
   * In order 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.
   * IMPORTANT: In order to encode using B frames, you must enable
   * frame reordering.
   */
  kICMCompressionSessionOptionsPropertyID_AllowFrameReordering = 'b ok', /* Boolean,
 Read/Write */
  /*
   * Indicates that durations of emitted frames are needed.
   * 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 clear, so frames will not be delayed in
   * order to calculate durations.
   * IMPORTANT: If you will be passing encoded frames to
   * AddMediaSampleFromEncodedFrame, you must set this flag to true.
   */
  kICMCompressionSessionOptionsPropertyID_DurationsNeeded = 'need', /* Boolean,
Read/Write */
  /*
   * The maximum interval between key frames, also known as the key
   * frame rate.
   * Key frames, also known as sync frames, reset inter-frame
   * dependencies; decoding a key frame is sufficient to prepare a
   * decompressor for correctly decoding the difference frames that
   * follow.
   * 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. A key
   * frame interval of 1 indicates that every frame must be a key
   * frame, 2 indicates that at least every other frame must be a key
   * frame, etc.
   */
  kICMCompressionSessionOptionsPropertyID_MaxKeyFrameInterval = 'kyfr', /* SInt32,
 Read/Write */
  /*
   * The requested maximum interval between partial sync frames. If the
   * interval is n, any sequence of n successive frames must include at
   * least one key or partial sync frame.
   * Where supported, partial sync frames perform a partial reset of
   * inter-frame dependencies; decoding two partial sync frames and the
   * non-droppable difference frames between them is sufficient to
   * prepare a decompressor for correctly decoding the difference
   * frames that follow.
   * Compressors are allowed to generate partial sync frames more
   * frequently if this would result in more efficient compression.
   *
   * The default partial sync frame interval is 0, which indicates that
   * the compressor should choose where to place partial sync frames. A
   * partial sync frame interval of 1 means there can be no difference
   * frames, so it is equivalent to a key frame interval of 1. A
   * partial sync frame interval of 2 means that every other frame must
   * be a key frame or a partial sync frame.
   * Compressors that do not support partial sync frames will ignore
   * this setting.
   */
  kICMCompressionSessionOptionsPropertyID_MaxPartialSyncFrameInterval = 'psfr', /*
 SInt32, Read/Write */
  /*
   * Enables the compressor to modify frame times.
   * Some compressors are able to identify and coalesce runs of
   * identical frames and output single frames with longer duration, 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 to preserve
   * frame display times.
   * (Note: this feature replaces the practice of having compressors
   * return special high similarity values to indicate that frames
   * could be dropped.)
   * If you want to allow the compressor to modify frame times in order
   * to improve compression performance, enable frame time changes.
   */
  kICMCompressionSessionOptionsPropertyID_AllowFrameTimeChanges = '+ ok', /* Boolean,
 Read/Write */
  /*
   * Enables the compressor to call the encoded-frame callback from a
   * different thread.
   * By default, the flag is false, which means that the compressor
   * must call the encoded-frame callback from the same thread that
   * ICMCompressionSessionEncodeFrame and
   * ICMCompressionSessionCompleteFrames were called on.
   */
  kICMCompressionSessionOptionsPropertyID_AllowAsyncCompletion = 'asok', /* Boolean,
 Read/Write */
  /*
   * 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. It 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 is kICMUnlimitedFrameDelayCount, which sets no limit
   * on the compression window.
   */
  kICMCompressionSessionOptionsPropertyID_MaxFrameDelayCount = 'cwin', /* SInt32,
 Read/Write */
  /*
   * The maximum frame delay time is the maximum difference between a
   * source frame's display time and the corresponding encoded frame's
   * decode time. It 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 is kICMUnlimitedFrameDelayTime, which sets no time
   * limit on the compression window.
   */
  kICMCompressionSessionOptionsPropertyID_MaxFrameDelayTime = 'cwit', /* TimeValue64,
 Read/Write */
```

```
  /*
   * Sets a specific compressor component or component instance to be
   * used, or one of the wildcards anyCodec, bestSpeedCodec,
   * bestFidelityCodec, or bestCompressionCodec.
   * Use this API to force the Image Compression Manager to use a
   * specific compressor component or compressor component instance.
   * (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.) To allow the Image Compression Manager to
   * choose the compressor component, set the compressorComponent to
   * anyCodec (the default), bestSpeedCodec, bestFidelityCodec or
   * bestCompressionCodec.
   */
  kICMCompressionSessionOptionsPropertyID_CompressorComponent = 'imco', /*
CompressorComponent, Read/Write */
  /*
   * A handle containing compressor settings. The compressor will be
   * configured with these settings (by a call to
   * ImageCodecSetSettings) during ICMCompressionSessionCreate.
   */
  kICMCompressionSessionOptionsPropertyID_CompressorSettings = 'cost', /* Handle,
 Read/Write */
  /*
   * 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_Depth = 'deep', /* UInt32, Read/Write */
  /*
   * The color table for compression.  Used with indexed-color depths.
   *
   * Clients who get this property are responsible for disposing the
   * returned CTabHandle.
   */
  kICMCompressionSessionOptionsPropertyID_ColorTable = 'clut', /* CTabHandle,
Read/Write*/
  /*
   * The compression quality.
   * This value is always used to set the spatialQuality; if temporal
   * compression is enabled, it is also used to set temporalQuality.
   * <BR> The default quality is codecNormalQuality.
   */
  kICMCompressionSessionOptionsPropertyID_Quality = 'qual', /* CodecQ, Read/Write
 */
  /*
   * The long-term desired average data rate in bytes per second.
   *  This is not a hard limit.
   * The default data rate is zero, which indicates that the quality
   * setting should determine the size of compressed data.
   * Note that data rate settings only have an effect when timing
   * information is provided for source frames, and that some codecs do
   * not support limiting to specified data rates.
   */
  kICMCompressionSessionOptionsPropertyID_AverageDataRate = 'aver', /* SInt32,
Read/Write */
  /*
   * 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, and 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).
   * Note that data rate settings only have an effect when timing
   * information is provided for source frames, and that some codecs do
   * not support limiting to specified data rates.
   */
  kICMCompressionSessionOptionsPropertyID_DataRateLimits = 'hard', /* C array of
ICMDataRateLimit struct, Read/Write */
  /*
   * The current number of data rate limits.
   */
  kICMCompressionSessionOptionsPropertyID_DataRateLimitCount = 'har#', /* UInt32,
 Read */
  /*
   * The maximum allowed number of data rate limits.  (Currently 2.)
   */
  kICMCompressionSessionOptionsPropertyID_MaxDataRateLimits = 'mhar', /* UInt32,
Read */
  /*
   * Indicates that the source was previously compressed.
   * This property is purely an optional, informational hint to the
   * compressor; by default it is false.
   */
  kICMCompressionSessionOptionsPropertyID_WasCompressed = 'wasc', /* Boolean,
Read/Write */
  /*
   * Recommends a CPU time budget for the 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 is an advisory hint to the compressor, and some compressors
   * may ignore it. Multithreaded compressors may use this amount of
   * CPU time on each processor.
   * Compressors should not feel compelled to use the full time budget
   * if they complete ahead of time!
   */
  kICMCompressionSessionOptionsPropertyID_CPUTimeBudget = 'cput', /* UInt32,
Read/Write */
  /*
   * Storage for multi-pass compression.
   * To enable multipass compression, the client must provide a storage
   * location for multipass data.  Use
   * ICMMultiPassStorageCreateWithTemporaryFile to have the ICM store
   * it in a temporary file.  Use
   * 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_MultiPassStorage = 'imps', /*
ICMMultiPassStorageRef, Read/Write */
  /*
   * Indicates the number of source frames, if known. If nonzero, this
   * should be 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_SourceFrameCount = 'frco', /* UInt64,
Read/Write */
  /*
   * Indicates the expected frame rate, if known. The frame rate is
   * measured in frames per second. This is not used to control the
   * frame rate; it is provided as a hint to the compressor so that it
   * can set up internal configuration before compression begins. The
   * actual frame rate will depend on frame durations and may vary. By
   * default, this is zero, indicating "unknown".
   */
  kICMCompressionSessionOptionsPropertyID_ExpectedFrameRate = 'fran', /* Fixed,
Read/Write */
  /*
   * Indicates how source frames to a compression session should be
   * scaled if the dimensions and/or display aspect ratio do not match.
   */
 kICMCompressionSessionOptionsPropertyID_ScalingMode = 'scam', /* OSType, Read/Write
 */
  /*
   * Describes the clean aperture for compressed frames. Note that if
   * the compressor enforces a clean aperture, it will override this
   * setting. The clean aperture will be set on the output image
   * description and may affect scaling in some scaling modes. By
   * default, this is all zeros, meaning unset.
   */
 kICMCompressionSessionOptionsPropertyID_CleanAperture = 'clap', /* Native-endian
 CleanApertureImageDescriptionExtension, Read/Write */
  /*
   * Describes the pixel aspect ratio for compressed frames. Note that
   * if the compressor enforces a pixel aspect ratio, it will override
   * this setting. The pixel aspect ratio will be set on the output
   * image description and may affect scaling in some scaling modes. By
   * default, this is all zeros, meaning unset.
   */
 kICMCompressionSessionOptionsPropertyID_PixelAspectRatio = 'pasp', /* Native-endian
 PixelAspectRatioImageDescriptionExtension, Read/Write */
  /*
   * Describes the number and order of fields for compressed frames.
   * Note that if the compressor enforces field info, it will override
   * this setting. The field info will be set on the output image
   * description and may affect scaling in some scaling modes. By
   * default, this is all zeros, meaning unset.
   */
  kICMCompressionSessionOptionsPropertyID_FieldInfo = 'fiel' /*
```

```
FieldInfoImageDescriptionExtension2, Read/Write */
};
enum {
  /*
   * Class identifier for compression session properties.
   */
  kQTPropertyClass_ICMCompressionSession = 'icse',
  /*
   * The time scale for the compression session.
   */
  kICMCompressionSessionPropertyID_TimeScale = 'tscl', /* TimeScale, Read */
  /*
   * The compressor's pixel buffer attributes for the compression
   * session. You can use these to create a pixel buffer pool for
   * source pixel buffers. Note that this is not the same as the
   * sourcePixelBufferAttributes passed in to
   * ICMCompressionSessionCreate. Getting this property does not change
   * its retain count.
   */
  kICMCompressionSessionPropertyID_CompressorPixelBufferAttributes = 'batt', /*
CFDictionaryRef, Read */
  /*
   * 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_PixelBufferPool = 'pool', /* CVPixelBufferPoolRef,
 Read */
  /*
   * The image description for the 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 this
   * handle when the compression session is disposed.
   * IMPORTANT: The caller must NOT dispose this handle.
   */
  kICMCompressionSessionPropertyID_ImageDescription = 'idsc' /*
ImageDescriptionHandle, Read */
};
enum {
  /*
   * Class identifier for decompression frame options object properties.
   */
  kQTPropertyClass_ICMDecompressionFrameOptions = 'idfo',
  /*
   * A specific pixel buffer that the frame should be decompressed
   * into. Setting this circumvents the pixel buffer pool mechanism. If
   * this buffer is not compatible with the codec's pixel buffer
   * requirements, decompression will fail.
   */
  kICMDecompressionFrameOptionsPropertyID_DestinationPixelBuffer = 'cvpb' /*
CVPixelBufferRef, Read/Write */
};
```

```
enum {
  /*
   * Class identifier for decompression session options object
   * properties.
   */
  kQTPropertyClass_ICMDecompressionSessionOptions = 'idso',
  /*
   * By default, this is true, meaning that frames must be output in
   * display order. Set this to false to allow frames to be output in
   * decode order rather than in display order.
   */
  kICMDecompressionSessionOptionsPropertyID_DisplayOrderRequired = 'dorq', /*
Boolean, Read/Write */
  /*
   * A specific decompressor component or component instance to be
   * used, or one of the wildcards anyCodec, bestSpeedCodec,
   * bestFidelityCodec, or bestCompressionCodec.
   * By default, this is anyCodec.
   */
  kICMDecompressionSessionOptionsPropertyID_DecompressorComponent = 'imdc', /*
DecompressorComponent, Read/Write */
  /*
   * The decompression accuracy.
   * The default accuracy is codecNormalQuality.
   */
 kICMDecompressionSessionOptionsPropertyID_Accuracy = 'acur', /* CodecQ, Read/Write
 */
  /*
   * Requests special handling of fields. Not all codecs will obey this
   * request; some codecs will only handle it at certain accuracy
   * levels. Ignored for non-interlaced content.
   */
  kICMDecompressionSessionOptionsPropertyID_FieldMode = 'fiel', /* ICMFieldMode,
Read/Write */
  /*
   * The maximum number of buffers ahead of the current time that
   * should be decompressed. Used in sessions that target visual
   * contexts. By default, the number of buffers will be determined
   * from the visual context.
   */
  kICMDecompressionSessionOptionsPropertyID_MaxBufferCount = 'm#bf', /* UInt32,
Read/Write */
  /*
   * The minimum time ahead of the current time that frames should be
   * decompressed. Used in sessions that target visual contexts. By
   * default, the output-ahead time will be determined from the visual
   * context.
   */
 kICMDecompressionSessionOptionsPropertyID_OutputAheadTime = 'futu' /* TimeRecord,
 Read/Write */
};
enum {
  /*
   * Class identifier for decompression session properties.
   */
  kQTPropertyClass_ICMDecompressionSession = 'icds',
  /*
   * The non-scheduled display time for a decompression session.
   * Setting this requests display of the non-scheduled queued frame at
   * that display time, if there is one.
   * See ICMDecompressionSessionSetNonScheduledDisplayTime.
   */
  kICMDecompressionSessionPropertyID_NonScheduledDisplayTime = 'nsti', /*
ICMNonScheduledDisplayTime, Read/Write */
  /*
   * The direction for non-scheduled display time.
   * See ICMDecompressionSessionSetNonScheduledDisplayDirection.
   */
  kICMDecompressionSessionPropertyID_NonScheduledDisplayDirection = 'nsdu', /*
Fixed, Read/Write */
  /*
   * The pixel buffer pool from which emitted pixel buffers are
   * allocated. Getting this does not change the retain count of the
   * pool.
   */
  kICMDecompressionSessionPropertyID_PixelBufferPool = 'pool', /*
CVPixelBufferPoolRef, Read */
  /*
   * Indicates whether the a common pixel buffer pool is shared between
   * the decompressor and the session client. This is false if separate
   * pools are used because the decompressor's and the client's pixel
   * buffer attributes were incompatible.
   */
  kICMDecompressionSessionPropertyID_PixelBufferPoolIsShared = 'plsh' /* Boolean,
 Read */
};
enum {
  /*
   * Class identifier for image description properties.
   */
  kQTPropertyClass_ImageDescription = 'idsc',
  /*
   * The width of the encoded image. Usually, but not always, this is
   * the ImageDescription's width field.
   */
  kICMImageDescriptionPropertyID_EncodedWidth = 'encw', /* SInt32, Read/Write */
  /*
   * The height of the encoded image. Usually, but not always, this is
   * the ImageDescription's height field.
   */
  kICMImageDescriptionPropertyID_EncodedHeight = 'ench', /* SInt32, Read/Write */
  /*
   * Describes the clean aperture of the buffer. If not specified
   * explicitly in the image description, the default clean aperture
   * (full encoded width and height) will be returned.
   */
  kICMImageDescriptionPropertyID_CleanAperture = 'clap', /* Native-endian
CleanApertureImageDescriptionExtension, Read/Write */
  /*
   * Describes the pixel aspect ratio. If not specified explicitly in
   * the image description, a square (1:1) pixel aspect ratio will be
   * returned.
   */
  kICMImageDescriptionPropertyID_PixelAspectRatio = 'pasp', /* Native-endian
PixelAspectRatioImageDescriptionExtension, Read/Write */
  /*
```

```
   * A width at which the buffer's image could be displayed on a
   * square-pixel display, possibly calculated using the clean aperture
   * and pixel aspect ratio.
   */
  kICMImageDescriptionPropertyID_DisplayWidth = 'disw', /* SInt32, Read */
  /*
   * A height at which the buffer's image could be displayed on a
   * square-pixel display, possibly calculated using the clean aperture
   * and pixel aspect ratio.
   */
  kICMImageDescriptionPropertyID_DisplayHeight = 'dish', /* SInt32, Read */
  /*
   * A width at which the image could be displayed on a square-pixel
   * display, disregarding any clean aperture but honoring the pixel
   * aspect ratio. This may be useful for authoring applications that
   * want to expose the edge processing region. For general viewing,
   * use kICMImageDescriptionPropertyID_DisplayWidth instead.
   */
  kICMImageDescriptionPropertyID_ProductionDisplayWidth = 'pdsw', /* SInt32, Read
 */
  /*
   * A height at which the image could be displayed on a square-pixel
   * display, disregarding any clean aperture but honoring the pixel
   * aspect ratio. This may be useful for authoring applications that
   * want to expose the edge processing region. For general viewing,
   * use kICMImageDescriptionPropertyID_DisplayHeight instead.
   */
  kICMImageDescriptionPropertyID_ProductionDisplayHeight = 'pdsh', /* SInt32, Read
 */
  /*
   * Color information, if available in the
   * NCLCColorInfoImageDescriptionExtension format.
   */
  kICMImageDescriptionPropertyID_NCLCColorInfo = 'nclc', /* Native-endian
NCLCColorInfoImageDescriptionExtension, Read/Write */
  /*
   * The gamma level described by the image description.
   */
  kICMImageDescriptionPropertyID_GammaLevel = 'gama', /* Fixed, Read/Write */
  /*
   * Information about the number and order of fields, if available.
   */
  kICMImageDescriptionPropertyID_FieldInfo = 'fiel', /*
FieldInfoImageDescriptionExtension2, Read/Write */
  /*
   * The offset in bytes from the start of one row to the next. Only
   * valid if the codec type is a chunky pixel format.
   */
  kICMImageDescriptionPropertyID_RowBytes = 'rowb', /* SInt32, Read/Write */
  /*
   * A track width suitable for passing to NewMovieTrack when creating
   * a new track to hold this image data.
   */
  kICMImageDescriptionPropertyID_ClassicTrackWidth = 'claw', /* Fixed, Read */
  /*
   * A track height suitable for passing to NewMovieTrack when creating
   * a new track to hold this image data.
   */
  kICMImageDescriptionPropertyID_ClassicTrackHeight = 'clah' /* Fixed, Read */
};
enum {
  /*
   * In this pass the compressor shall output encoded frames.
   */
  kICMCompressionPassMode_OutputEncodedFrames = 1L << 0,
  /*
   * In this pass the client need not provide source frame buffers.
   */
  kICMCompressionPassMode_NoSourceFrames = 1L << 1,
  /*
   * In this pass the compressor may write private data to multipass
   * storage.
   */
  kICMCompressionPassMode_WriteToMultiPassStorage = 1L << 2,
  /*
   * In this pass the compressor may read private data from multipass
   * storage.
   */
  kICMCompressionPassMode_ReadFromMultiPassStorage = 1L << 3,
  /*
   * The compressor will set this flag to indicate that it will not be
   * able to output encoded frames in the coming pass. If this flag is
   * not set, then the client is allowed to set the
   * kICMCompressionPassMode_OutputEncodedFrames flag before calling
   * ICMCompressionSessionBeginPass.
   */
  kICMCompressionPassMode_NotReadyToOutputEncodedFrames = 1L << 4
};
enum {
  /*
   * Indicates that this is the last call for this sourceFrameRefCon.
   */
  kICMSourceTracking_LastCall   = 1L << 0,
  /*
   * Indicates that the session is done with the source pixel buffer
   * and has released any reference to it that it had.
   */
  kICMSourceTracking_ReleasedPixelBuffer = 1L << 1,
  /*
   * Indicates that this frame was encoded.
   */
  kICMSourceTracking_FrameWasEncoded = 1L << 2,
  /*
   * Indicates that this frame was dropped.
   */
  kICMSourceTracking_FrameWasDropped = 1L << 3,
  /*
   * Indicates that this frame was merged into other frames.
   */
  kICMSourceTracking_FrameWasMerged = 1L << 4,
  /*
   * Indicates that the time stamp of this frame was modified.
   */
  kICMSourceTracking_FrameTimeWasChanged = 1L << 5,
  /*
   * Indicates that the ICM has copied the image from the source pixel
   * buffer into another pixel buffer because the source pixel buffer
   * was not compatible with the compressor's required pixel buffer
   * attributes.
   */
  kICMSourceTracking_CopiedPixelBuffer = 1L << 6
};
enum {
  /*
   * The full width and height of source frames shall be scaled to the
   * full width and height of the destination. This is the default if
   * no other scaling mode is specified.
   */
  kICMScalingMode_StretchProductionAperture = 'sp2p',
  /*
   * The clean aperture of the source frames shall be scaled to the
   * clean aperture of the destination.
   */
  kICMScalingMode_StretchCleanAperture = 'sc2c',
  /*
   * The clean aperture of the source frames shall be scaled to fit
   * inside the clean aperture of the destination, preserving the
   * original display aspect ratio. If the display aspect ratios are
   * different, the source frames will be centered with black bars
   * above and below, or to the left and right.
   */
  kICMScalingMode_Letterbox     = 'lett',
  /*
   * The clean aperture of the source frames shall be scaled to cover
   * the clean aperture of the destination, preserving the original
   * display aspect ratio. If the display aspect ratios are different,
   * the source frames will be centered and cropped.
   */
  kICMScalingMode_Trim          = 'trim'
};
```

Constants

`kICMCompressionFrameOptionsPropertyID_ForceKeyFrame`

Boolean, ReadWrite.

Available in Mac OS X v10.3 and later.

Declared in `ImageCompression.h`.

`kICMCompressionFrameOptionsPropertyID_FrameType`

`ICMFrameType`, ReadWrite.

Available in Mac OS X v10.3 and later.

Declared in `ImageCompression.h`.

`kQTPropertyClass_ICMCompressionSessionOptions`

Class identifier for compression session option object properties. Also `'icso'`.

Available in Mac OS X v10.3 and later.

Declared in `ImageCompression.h`.

`kICMCompressionSessionOptionsPropertyID_AllowTemporalCompression`

Enables temporal compression of P-frames and B-frames. By default, temporal compression is disabled. Also `'p ok'`.

Available in Mac OS X v10.3 and later.

Declared in `ImageCompression.h`.

`kICMCompressionSessionOptionsPropertyID_AllowFrameReordering`
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. Also `'b ok'`.

Available in Mac OS X v10.3 and later.

Declared in `ImageCompression.h`.

`kICMCompressionSessionOptionsPropertyID_DurationsNeeded`
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. Also `'need'`.

Available in Mac OS X v10.3 and later.

Declared in `ImageCompression.h`.

`kICMCompressionSessionOptionsPropertyID_MaxKeyFrameInterval`
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. Also `'kyfr'`.

Available in Mac OS X v10.3 and later.

Declared in `ImageCompression.h`.

`kICMCompressionSessionOptionsPropertyID_MaxPartialSyncFrameInterval`
SInt32, ReadWrite.

Available in Mac OS X v10.3 and later.

Declared in `ImageCompression.h`.

`kICMCompressionSessionOptionsPropertyID_AllowFrameTimeChanges`
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. Also `'+ ok'`.

Available in Mac OS X v10.3 and later.

Declared in `ImageCompression.h`.

`kICMCompressionSessionOptionsPropertyID_AllowAsyncCompletion`
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`. Also `'asok'`.

Available in Mac OS X v10.3 and later.

Declared in `ImageCompression.h`.

`kICMCompressionSessionOptionsPropertyID_MaxFrameDelayCount`
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. Also `'cwin'`.

Available in Mac OS X v10.3 and later.

Declared in `ImageCompression.h`.

`kICMCompressionSessionOptionsPropertyID_MaxFrameDelayTime`
TimeValue64, ReadWrite.

Available in Mac OS X v10.3 and later.

Declared in `ImageCompression.h`.

`kICMCompressionSessionOptionsPropertyID_CompressorComponent`
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. Also `'imco'`.

Available in Mac OS X v10.3 and later.

Declared in `ImageCompression.h`.

`kICMCompressionSessionOptionsPropertyID_CompressorSettings`
A handle containing compressor settings. The compressor will be configured with these settings (by a call to `ImageCodecSetSettings`) during the `ICMCompressionSessionCreate` process. Also `'cost'`.

Available in Mac OS X v10.3 and later.

Declared in `ImageCompression.h`.

`kICMCompressionSessionOptionsPropertyID_Depth`
UInt32, ReadWrite.

Available in Mac OS X v10.3 and later.

Declared in `ImageCompression.h`.

`kICMCompressionSessionOptionsPropertyID_ColorTable`
The color table for compression, used with indexed-color depths. Clients who are passed this property are responsible for disposing the returned `CTabHandle`. Also `'clut'`.

Available in Mac OS X v10.3 and later.

Declared in `ImageCompression.h`.

`kICMCompressionSessionOptionsPropertyID_Quality`
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`. Also `'qual'`.

Available in Mac OS X v10.3 and later.

Declared in `ImageCompression.h`.

`kICMCompressionSessionOptionsPropertyID_AverageDataRate`
: 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. Also `'aver'`.

  Available in Mac OS X v10.3 and later.

  Declared in `ImageCompression.h`.

`kICMCompressionSessionOptionsPropertyID_DataRateLimits`
: 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. Also `'hard'`.

  Available in Mac OS X v10.3 and later.

  Declared in `ImageCompression.h`.

`kICMCompressionSessionOptionsPropertyID_DataRateLimitCount`
: UInt32, Read.

  Available in Mac OS X v10.3 and later.

  Declared in `ImageCompression.h`.

`kICMCompressionSessionOptionsPropertyID_MaxDataRateLimits`
: UInt32, Read.

  Available in Mac OS X v10.3 and later.

  Declared in `ImageCompression.h`.

`kICMCompressionSessionOptionsPropertyID_WasCompressed`
: Indicates that the source was previously compressed. This property is an optional information hint to the compressor; by default it is FALSE. Also `'wasc'`.

  Available in Mac OS X v10.3 and later.

  Declared in `ImageCompression.h`.

`kICMCompressionSessionOptionsPropertyID_CPUTimeBudget`
: UInt32, ReadWrite.

  Available in Mac OS X v10.3 and later.

  Declared in `ImageCompression.h`.

`kICMCompressionSessionOptionsPropertyID_MultiPassStorage`
: 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. Also `'imps'`.

  Available in Mac OS X v10.3 and later.

  Declared in `ImageCompression.h`.

`kICMCompressionSessionOptionsPropertyID_SourceFrameCount`
: UInt64, ReadWrite.

  Available in Mac OS X v10.3 and later.

  Declared in `ImageCompression.h`.

`kICMCompressionSessionOptionsPropertyID_ExpectedFrameRate`
: `Fixed`, ReadWrite.

  Available in Mac OS X v10.3 and later.

  Declared in `ImageCompression.h`.

`kICMCompressionSessionOptionsPropertyID_ScalingMode`
: `OSType`, ReadWrite.

  Available in Mac OS X v10.3 and later.

  Declared in `ImageCompression.h`.

`kICMCompressionSessionOptionsPropertyID_CleanAperture`
: Native-endian `CleanApertureImageDescriptionExtension`, ReadWrite.

  Available in Mac OS X v10.3 and later.

  Declared in `ImageCompression.h`.

`kICMCompressionSessionOptionsPropertyID_PixelAspectRatio`
: Native-endian `PixelAspectRatioImageDescriptionExtension`, ReadWrite.

  Available in Mac OS X v10.3 and later.

  Declared in `ImageCompression.h`.

`kICMCompressionSessionOptionsPropertyID_FieldInfo`
: FieldInfoImageDescriptionExtension2, ReadWrite.

  Available in Mac OS X v10.3 and later.

  Declared in `ImageCompression.h`.

`kQTPropertyClass_ICMCompressionSession`
: Class identifier for compression session properties. Also `'icse'`.

  Available in Mac OS X v10.3 and later.

  Declared in `ImageCompression.h`.

`kICMCompressionSessionPropertyID_TimeScale`
: The time scale for the compression session. Also `'tscl'`.

  Available in Mac OS X v10.3 and later.

  Declared in `ImageCompression.h`.

`kICMCompressionSessionPropertyID_CompressorPixelBufferAttributes`
: 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. Also `'batt'`.

  Available in Mac OS X v10.3 and later.

  Declared in `ImageCompression.h`.

`kICMCompressionSessionPropertyID_PixelBufferPool`
: 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. Also `'pool'`.

  Available in Mac OS X v10.3 and later.

  Declared in `ImageCompression.h`.

`kICMCompressionSessionPropertyID_ImageDescription`
: 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. Also `'idsc'`.

  Available in Mac OS X v10.3 and later.

  Declared in `ImageCompression.h`.

`kICMDecompressionFrameOptionsPropertyID_DestinationPixelBuffer`
: `CVPixelBufferRef`, ReadWrite.

  Available in Mac OS X v10.3 and later.

  Declared in `ImageCompression.h`.

`kICMDecompressionSessionOptionsPropertyID_DisplayOrderRequired`
: Boolean, ReadWrite.

  Available in Mac OS X v10.3 and later.

  Declared in `ImageCompression.h`.

`kICMDecompressionSessionOptionsPropertyID_DecompressorComponent`
: `DecompressorComponent`, ReadWrite.

  Available in Mac OS X v10.3 and later.

  Declared in `ImageCompression.h`.

`kICMDecompressionSessionOptionsPropertyID_Accuracy`
: `CodecQ`, ReadWrite.

  Available in Mac OS X v10.3 and later.

  Declared in `ImageCompression.h`.

`kICMDecompressionSessionOptionsPropertyID_FieldMode`
: `ICMFieldMode`, ReadWrite.

  Available in Mac OS X v10.3 and later.

  Declared in `ImageCompression.h`.

`kICMDecompressionSessionOptionsPropertyID_MaxBufferCount`
UInt32, ReadWrite.

Available in Mac OS X v10.3 and later.

Declared in `ImageCompression.h`.

`kICMDecompressionSessionOptionsPropertyID_OutputAheadTime`
`TimeRecord`, ReadWrite.

Available in Mac OS X v10.3 and later.

Declared in `ImageCompression.h`.

`kICMDecompressionSessionPropertyID_NonScheduledDisplayTime`
`ICMNonScheduledDisplayTime`, ReadWrite.

Available in Mac OS X v10.3 and later.

Declared in `ImageCompression.h`.

`kICMDecompressionSessionPropertyID_NonScheduledDisplayDirection`
`Fixed`, ReadWrite.

Available in Mac OS X v10.3 and later.

Declared in `ImageCompression.h`.

`kICMDecompressionSessionPropertyID_PixelBufferPool`
`CVPixelBufferPoolRef`, Read.

Available in Mac OS X v10.3 and later.

Declared in `ImageCompression.h`.

`kICMDecompressionSessionPropertyID_PixelBufferPoolIsShared`
Boolean, Read.

Available in Mac OS X v10.3 and later.

Declared in `ImageCompression.h`.

`kICMImageDescriptionPropertyID_EncodedWidth`
SInt32, ReadWrite.

Available in Mac OS X v10.3 and later.

Declared in `ImageCompression.h`.

`kICMImageDescriptionPropertyID_EncodedHeight`
SInt32, ReadWrite.

Available in Mac OS X v10.3 and later.

Declared in `ImageCompression.h`.

`kICMImageDescriptionPropertyID_CleanAperture`
Native-endian `CleanApertureImageDescriptionExtension`, ReadWrite.

Available in Mac OS X v10.3 and later.

Declared in `ImageCompression.h`.

`kICMImageDescriptionPropertyID_PixelAspectRatio`
Native-endian `PixelAspectRatioImageDescriptionExtension`, ReadWrite.

Available in Mac OS X v10.3 and later.

Declared in `ImageCompression.h`.

`kICMImageDescriptionPropertyID_DisplayWidth`
SInt32, Read.

Available in Mac OS X v10.3 and later.

Declared in `ImageCompression.h`.

`kICMImageDescriptionPropertyID_DisplayHeight`
SInt32, Read.

Available in Mac OS X v10.3 and later.

Declared in `ImageCompression.h`.

`kICMImageDescriptionPropertyID_ProductionDisplayWidth`
SInt32, Read.

Available in Mac OS X v10.3 and later.

Declared in `ImageCompression.h`.

`kICMImageDescriptionPropertyID_ProductionDisplayHeight`
SInt32, Read.

Available in Mac OS X v10.3 and later.

Declared in `ImageCompression.h`.

`kICMImageDescriptionPropertyID_NCLCColorInfo`
Native-endian `NCLCColorInfoImageDescriptionExtension`, ReadWrite.

Available in Mac OS X v10.3 and later.

Declared in `ImageCompression.h`.

`kICMImageDescriptionPropertyID_GammaLevel`
`Fixed`, ReadWrite.

Available in Mac OS X v10.3 and later.

Declared in `ImageCompression.h`.

`kICMImageDescriptionPropertyID_FieldInfo`
FieldInfoImageDescriptionExtension2, ReadWrite.

Available in Mac OS X v10.3 and later.

Declared in `ImageCompression.h`.

`kICMImageDescriptionPropertyID_RowBytes`
SInt32, ReadWrite.

Available in Mac OS X v10.3 and later.

Declared in `ImageCompression.h`.

`kICMImageDescriptionPropertyID_ClassicTrackWidth`
`Fixed`, Read.

Available in Mac OS X v10.3 and later.

Declared in `ImageCompression.h`.

`kICMImageDescriptionPropertyID_ClassicTrackHeight`
`Fixed`, Read.

Available in Mac OS X v10.3 and later.

Declared in `ImageCompression.h`.

Declared In
`ImageCompression.h`

ICMEncodedFrameSetFrameType Values

Constants passed to ICMEncodedFrameSetFrameType.

```
enum {
  kICMFrameType_I               = 'I',
  kICMFrameType_P               = 'P',
  kICMFrameType_B               = 'B',
  kICMFrameType_Unknown         = 0
};
```

Declared In
`ImageCompression.h`

ICMMultiPassStorageCreateWithTemporaryFile Values

Constants passed to ICMMultiPassStorageCreateWithTemporaryFile.

```
enum {
  /*
   * Indicates that the temporary file should not be deleted when the
   * multipass storage is released.
   */
  kICMMultiPassStorage_DoNotDeleteWhenDone = 1L << 0
};
```

Declared In
`ImageCompression.h`

ICMMultiPassStorageGetTimeStamp Values

Constants passed to ICMMultiPassStorageGetTimeStamp.

```
enum {
  /*
   * Requests the first time stamp at which a value is stored.
   */
  kICMMultiPassStorage_GetFirstTimeStamp = 1,
  /*
   * Requests the previous time stamp before the given time stamp at
   * which a value is stored.
   */
  kICMMultiPassStorage_GetPreviousTimeStamp = 2,
  /*
   * Requests the next time stamp after the given time stamp at which a
   * value is stored.
   */
  kICMMultiPassStorage_GetNextTimeStamp = 3,
  /*
   * Requests the last time stamp at which a value is stored.
   */
  kICMMultiPassStorage_GetLastTimeStamp = 4
};
```

Declared In
`ImageCompression.h`

kICMValidTime_DecodeDurationIsValid

Constants grouped with kICMValidTime_DecodeDurationIsValid.

```
enum {
  /*
   * Indicates that a display time stamp is valid.
   */
  kICMValidTime_DisplayTimeStampIsValid = 1L << 0,
  /*
   * Indicates that a display duration is valid.
   */
  kICMValidTime_DisplayDurationIsValid = 1L << 1,
  /*
   * Indicates that a decode time stamp is valid.
   */
  kICMValidTime_DecodeTimeStampIsValid = 1L << 2,
  /*
   * Indicates that a decode duration is valid.
   */
  kICMValidTime_DecodeDurationIsValid = 1L << 3,
  /*
   * Indicates that a display offset (the offset from a decode time
   * stamp to a display time stamp) is valid.
   */
  kICMValidTime_DisplayOffsetIsValid = 1L << 4
};
```

Constants

`kICMValidTime_DisplayTimeStampIsValid`

The time value passed in `displayTimeStamp` is valid.

Available in Mac OS X v10.3 and later.

Declared in `ImageCompression.h`.

`kICMValidTime_DisplayDurationIsValid`

The time value passed in `displayDuration` is valid.

Available in Mac OS X v10.3 and later.

Declared in `ImageCompression.h`.

Declared In

`ImageCompression.h`

Movie Manager Reference

Framework:	Frameworks/QuickTime.framework
Declared in	Movies.h

Overview

QuickTime movies have certain overall timing and other presentation characteristics that an application can manage, including the presentation of special kinds of media such as flash media and sprites.

Functions by Task

Controlling Movie Playback

`GoToBeginningOfMovie` (page 240)
Repositions a movie to play from its start.

`GoToEndOfMovie` (page 241)
Repositions a movie to play from its end.

`StartMovie` (page 332)
Starts the movie playing from the current movie time.

`StopMovie` (page 333)
Stops the playback of a movie.

Creating and Disposing of Time Bases

`ChooseMovieClock` (page 181)
Searches media handlers to find the best clock for a movie.

`DisposeTimeBase` (page 189)
Disposes of a time base once you are finished with it.

`GetTimeBaseMasterClock` (page 229)
Determines the clock component that is assigned to a time base.

`GetTimeBaseMasterTimeBase` (page 231)
Determines the master time base that is assigned to a time base.

`NewTimeBase` (page 261)
Obtains a new time base.

`QTGetWallClockTimeBase` (page 271)
Returns the system's real-time time base.

`SetMovieMasterClock` (page 290)
Assigns a clock component to a movie.

`SetMovieMasterTimeBase` (page 291)
Assigns a master time base to a movie.

`SetTimeBaseMasterClock` (page 304)
Assigns a clock component to a time base.

`SetTimeBaseMasterTimeBase` (page 305)
Assigns a master time base to a time base.

`SetTimeBaseZero` (page 309)
Changes the offset from a time base to either its master time base or its clock component.

Determining Movie Creation and Modification Time

`GetMovieCreationTime` (page 209)
Returns the movie's creation date and time information.

`GetMovieModificationTime` (page 213)
Returns a movie's modification date and time.

Disabling Movies and Tracks

`GetMovieActive` (page 204)
Determines whether a movie is currently active.

`SetMovieActive` (page 284)
Activates or deactivates a movie.

Enhancing Movie Playback Performance

`AbortPrePrerollMovie` (page 177)
Terminates the operation of PrePrerollMovie.

`GetMovieActiveSegment` (page 205)
Determines what portion of a movie is currently active for playing.

`LoadMediaIntoRam` (page 245)
Loads a media's data into memory.

`LoadMovieIntoRam` (page 246)
Loads a movie's data into memory.

`LoadTrackIntoRam` (page 247)
Loads a track's data into memory.

`PrePrerollMovie` (page 263)
Sets up any necessary network connections to receive streaming content.

`PrerollMovie` (page 264)
Prepares a portion of a movie for playback.

`SetMovieActiveSegment` (page 285)
Defines a movie's active segment.

Error Functions

`ClearMoviesStickyError` (page 182)
Clears the sticky error value.

`GetMoviesError` (page 221)
Returns the contents of the current error value and resets the current error value to 0.

`GetMoviesStickyError` (page 222)
Returns the contents of the sticky error value.

`SetMoviesErrorProc` (page 298)
Performs custom error notification.

Generating Pictures From Movies

`GetMoviePict` (page 214)
Creates a QuickDraw picture from a specified movie at a specified time.

`GetMoviePosterPict` (page 215)
Creates a QuickDraw picture that contains a movie's poster.

`GetTrackPict` (page 239)
Creates a QuickDraw picture from a specified track at a specified time.

High-Level Movie Editing Functions

`GetMovieSelection` (page 220)
Returns information about a movie's current selection.

`SetMovieSelection` (page 297)
Sets a movie's current selection.

Initializing the Movie Toolbox

`EnterMovies` (page 189)
Initializes the Movie Toolbox and creates a private storage area for your application.

`ExitMovies` (page 192)
Automatically called when an application quits.

Managing Movie Sprites

`SpriteMediaCountImages` (page 313)
Retrieves the number of images that currently exist in a sprite track.

`SpriteMediaCountSprites` (page 314)
: Retrieves the number of sprites that currently exist in a sprite track.

`SpriteMediaGetDisplayedSampleNumber` (page 317)
: Retrieves the number of the sprite media sample that is currently being displayed.

`SpriteMediaGetImageName` (page 317)
: Returns the name of the image with the specified index from the current key frame sample.

`SpriteMediaGetIndImageDescription` (page 318)
: Retrieves an image description for a specified image in a sprite track.

`SpriteMediaGetIndImageProperty` (page 319)
: Returns a property value for a sprite image specified by an index.

`SpriteMediaGetSpriteName` (page 321)
: Returns the name of the sprite with the specified ID from the currently displayed sample.

`SpriteMediaGetSpriteProperty` (page 322)
: Retrieves the value of the specified sprite or sprite track property.

`SpriteMediaHitTestAllSprites` (page 323)
: Determines whether any sprites are at a specified location.

`SpriteMediaHitTestOneSprite` (page 324)
: Performs a hit testing operation on the sprite specified by a spriteID.

`SpriteMediaSetSpriteProperty` (page 330)
: Sets the specified property of a sprite or sprite track.

`SpriteMediaSpriteIDToIndex` (page 331)
: Converts a sprite ID to the corresponding sprite index.

`SpriteMediaSpriteIndexToID` (page 331)
: Returns the ID of a sprite specified by a sprite index.

Managing Sprite Images Outside a Movie

`SpriteMediaDisposeImage` (page 314)
: Frees the memory allocated for a sprite image outside a movie and removes that image from the sprite track in which it appears.

`SpriteMediaImageIDToIndex` (page 325)
: Returns the index of an outside sprite image from the ID of that image.

`SpriteMediaImageIndexToID` (page 326)
: Returns the ID of an outside sprite image from the index of that image.

`SpriteMediaNewImage` (page 326)
: Creates a new movie sprite image outside a movie.

Managing the Video Frame Playback Rate

`VideoMediaGetStatistics` (page 349)
: Returns the play-back frame rate of a movie.

`VideoMediaResetStatistics` (page 350)
: Resets the video media handler's counters before using VideoMediaGetStatistics to determine the frame rate of a movie.

Movie Functions

`DisposeMovie` (page 188)
: Frees any memory being used by a movie, including the memory used by the movie's tracks and media structures.

`NewMovie` (page 259)
: Creates a new movie in memory.

`PutMovieForDataRefIntoHandle` (page 265)
: Puts a self-contained movie into a handle.

Movie Posters and Movie Previews

`GetMoviePosterTime` (page 216)
: Returns the poster's time in a movie.

`GetMoviePreviewMode` (page 218)
: Determines whether a movie is in preview mode.

`GetMoviePreviewTime` (page 218)
: Returns the starting time and duration of the movie's preview.

`PlayMoviePreview` (page 262)
: Plays a movie's preview.

`SetMoviePosterTime` (page 293)
: Sets the poster time for the movie.

`SetMoviePreviewMode` (page 295)
: Places a movie into and out of preview mode.

`SetMoviePreviewTime` (page 296)
: Defines the starting time and duration of the movie's preview.

`ShowMoviePoster` (page 312)
: Displays a movie's poster.

Movie Toolbox Clock Support Functions

`AddCallBackToTimeBase` (page 178)
: Places a callback event into the list of scheduled callback events.

`ExecuteCallBack` (page 192)
: Called by a clock component when it determines that it is time to execute a callback function.

`GetFirstCallBack` (page 204)
: Returns the first callback event associated with a specified time base.

`GetNextCallBack` (page 227)
: Returns the next callback event associated with a specified time base.

`RemoveCallBackFromTimeBase` (page 284)
: Removes a callback event from the list of scheduled callback events.

Movies and Your Event Loop

`CreateMovieControl` (page 184)
Creates a movie control object to pass to the Mac OS Control Manager.

`InvalidateMovieRegion` (page 241)
Invalidates a small area of a movie.

`IsMovieDone` (page 242)
Determines if a particular movie has completely finished playing.

`MoviesTask` (page 257)
Services active movies.

`QTGetTimeUntilNextTask` (page 270)
Reports the duration until the next time QuickTime needs to run a task.

`QTInstallNextTaskNeededSoonerCallback` (page 277)
Installs a QTNextTaskNeededSoonerCallbackProc callback.

`QTUninstallNextTaskNeededSoonerCallback` (page 283)
Removes a QTNextTaskNeededSoonerCallbackProc callback.

`UpdateMovie` (page 347)
Ensures that the Movie Toolbox properly displays your movie after it has been uncovered.

Preferred Movie Settings

`GetMoviePreferredRate` (page 217)
Returns a movie's default playback rate.

`GetMoviePreferredVolume` (page 217)
Returns a movie's preferred volume setting.

`SetMoviePreferredRate` (page 293)
Specifies a movie's default playback rate.

`SetMoviePreferredVolume` (page 294)
Sets a movie's preferred volume setting.

Saving Movies

`PutMovieIntoDataFork` (page 266)
Stores a movie in the data fork of a given file.

`PutMovieIntoHandle` (page 268)
Creates a new movie resource.

`PutMovieIntoStorage` (page 268)
Writes a movie to a storage location managed by a data handler.

Text Media Handler Functions

`TextMediaAddHiliteSample` (page 334)
Provides dynamic highlighting of text.

`TextMediaAddTESample` (page 335)
: Specifies a TextEdit handle to be added to a specified media.

`TextMediaAddTextSample` (page 337)
: Adds a single block of styled text to an existing media.

`TextMediaFindNextText` (page 340)
: Searches for text with a specified media handler starting at a given time.

`TextMediaHiliteTextSample` (page 342)
: Specifies selected text to be highlighted for a given text media handler.

`TextMediaSetTextProc` (page 345)
: Specifies a custom function to be called whenever a text sample is displayed in a movie.

`TextMediaSetTextSampleData` (page 346)
: Sets values before calling TextMediaAddTextSample or TextMediaAddTESample.

The Sound Description Structure

`QTSoundDescriptionConvert` (page 278)
: Converts a sound description from one version to another.

Time Base Callback Functions

`CallMeWhen` (page 179)
: Schedules a callback event.

`CancelCallBack` (page 180)
: Cancels a callback event before it executes.

`DisposeCallBack` (page 186)
: Disposes of a callback event.

`GetCallBackTimeBase` (page 203)
: Retrieves the time base of a callback event.

`GetCallBackType` (page 203)
: Retrieves a callback event's type.

`NewCallBack` (page 258)
: Creates a new callback event.

Using the OpenGL Texture Context

`GetMovieVisualContext` (page 226)
: Returns the current visual context for a movie.

`SetMovieVisualContext` (page 302)
: Targets a movie to render into a visual context.

Working With Movie Spatial Characteristics

`DisposeMatte` (page 187)
    Disposes of a matte obtained from the GetTrackMatte function.

`GetMovieBoundsRgn` (page 206)
    Determines a movie's boundary region.

`GetMovieBox` (page 207)
    Returns a movie's boundary rectangle, which is a rectangle that encompasses all of the movie's enabled tracks.

`GetMovieClipRgn` (page 208)
    Determines a movie's clipping region.

`GetMovieDisplayBoundsRgn` (page 209)
    Determines a movie's display boundary region.

`GetMovieDisplayClipRgn` (page 210)
    Determines a movie's current display clipping region.

`GetMovieGWorld` (page 212)
    Returns a movie's graphics world.

`GetMovieMatrix` (page 213)
    Retrieves a movie's transformation matrix.

`GetTrackBoundsRgn` (page 236)
    Lets the media limit the size of a track boundary rectangle.

`GetTrackClipRgn` (page 237)
    Determines the clipping region of a track.

`GetTrackDisplayBoundsRgn` (page 237)
    Determines the region a track occupies in a movie's graphics world.

`GetTrackMatte` (page 238)
    Retrieves a copy of a track's matte.

`GetTrackMovieBoundsRgn` (page 239)
    Determines the region the track occupies in a movie's boundary region.

`SetMovieBox` (page 286)
    Sets a movie's boundary rectangle.

`SetMovieClipRgn` (page 287)
    Establishes a movie's clipping region.

`SetMovieDisplayClipRgn` (page 288)
    Establishes a movie's current display clipping region.

`SetMovieGWorld` (page 290)
    Establishes a movie's display coordinate system by setting the graphics world for displaying the movie.

`SetMovieMatrix` (page 292)
    Sets a movie's transformation matrix.

`SetTrackClipRgn` (page 310)
    Sets the clipping region of a track.

`SetTrackGWorld` (page 311)
    Forces a track to draw into a particular graphics world, which may be different from that of the movie.

SetTrackMatte (page 312)
Sets a track's matte.

Working With Movie Time

GetMovieDuration (page 211)
Returns the duration of a movie.

GetMovieRate (page 219)
Returns a movie's playback rate.

GetMovieTime (page 223)
Returns a movie's current time both as a time value and in a time structure.

GetMovieTimeBase (page 224)
Returns a movie's time base.

GetMovieTimeScale (page 224)
Returns the time scale of a movie.

SetMovieRate (page 296)
Sets a movie's playback rate.

SetMovieTime (page 299)
Changes a movie's current time.

SetMovieTimeScale (page 300)
Establishes a movie's time scale.

SetMovieTimeValue (page 300)
Sets a movie's time value.

Working With Progress and Cover Functions

SetMovieDrawingCompleteProc (page 289)
Assigns a drawing-complete function to a movie.

Working With Sound Descriptions

QTSoundDescriptionCreate (page 279)
Creates a sound description structure of the requested kind from an AudioStreamBasicDescription, optional audio channel layout, and optional magic cookie.

Working With Sound Volume

GetMovieVolume (page 226)
Returns a movie's current volume setting.

SetMovieVolume (page 302)
Sets a movie's current volume but does not store the setting in the movie.

Working With The Idle Manager

QTIdleManagerClose (page 271)
Closes the Mac OS Idle Manager.

QTIdleManagerGetNextIdleTime (page 272)
Retrieves the next idle time known to the Idle Manager.

QTIdleManagerNeedsAnIdle (page 272)
Tells the Idle Manager whether an idle will be required.

QTIdleManagerOpen (page 273)
Opens the Mac OS Idle Manager.

QTIdleManagerSetNextIdleTime (page 273)
Informs the idle manager of the next required idle time.

QTIdleManagerSetNextIdleTimeDelta (page 274)
Informs the idle manager of the time from the currently set idle time to the next idle time required after it.

QTIdleManagerSetNextIdleTimeNever (page 275)
Sets the next idle time indefinitely in the future.

QTIdleManagerSetNextIdleTimeNow (page 275)
Requests an idle as soon as possible.

QTIdleManagerSetParent (page 276)
Sets the parent of an Idle Manager instance.

Working With Time Base Values

GetTimeBaseEffectiveRate (page 228)
Returns the effective rate at which the specified time base is moving relative to its master clock.

GetTimeBaseFlags (page 229)
Obtains the control flags of a time base.

GetTimeBaseRate (page 231)
Retrieves the rate of a time base.

GetTimeBaseStartTime (page 233)
Determines the start time of a time base.

GetTimeBaseStatus (page 234)
Determines when the current time of a time base would fall outside of the range of values specified by the time base's start and stop times.

GetTimeBaseStopTime (page 234)
Determines the stop time of a time base.

GetTimeBaseTime (page 235)
Obtains the current time value from a time base.

SetTimeBaseFlags (page 303)
Sets the contents of the control flags of a time base.

SetTimeBaseRate (page 306)
Sets the rate of a time base.

`SetTimeBaseStartTime` (page 307)
Sets the start time of a time base.

`SetTimeBaseStopTime` (page 307)
Sets the stop time of a time base.

`SetTimeBaseTime` (page 308)
Sets the current time of a time base.

`SetTimeBaseValue` (page 309)
Sets the current time of a time base.

Working With Times

`AddTime` (page 178)
Adds two times.

`ConvertTime` (page 183)
Converts a time obtained from one time base into a time that is relative to another time base.

`ConvertTimeScale` (page 183)
Converts a time from one time scale into a time that is relative to another time scale.

`SubtractTime` (page 333)
Subtracts one time from another.

Working With User Data

`GetMovieUserData` (page 225)
Obtains access to a movie's user data list.

Working With Wired Sprites

`SpriteMediaGetActionVariable` (page 315)
Returns the value of the sprite track variable with the specified ID.

`SpriteMediaSetActionVariable` (page 328)
Sets the value of a sprite track variable to a specified value.

Supporting Functions

`AttachTimeBaseToCurrentThread` (page 179)
Attaches a time base to the current thread.

`CheckQuickTimeRegistration` (page 181)
Deprecated.

`ConvertTimeToClockTime` (page 184)
Converts a time record in a time base to clock time.

`DetachTimeBaseFromCurrentThread` (page 186)
Detaches a time base from the current thread.

`EnterMoviesOnThread` (page 191)
: Indicates that the client will be using QuickTime on the current thread.

`ExitMoviesOnThread` (page 194)
: Indicates to QuickTime that the client will no longer be using QuickTime on the current thread.

`FlashMediaDoButtonActions` (page 194)
: Performs actions attached to a specified button.

`FlashMediaFrameLabelToMovieTime` (page 195)
: Undocumented

`FlashMediaFrameNumberToMovieTime` (page 196)
: Undocumented

`FlashMediaGetDisplayedFrameNumber` (page 196)
: Undocumented

`FlashMediaGetFlashVariable` (page 197)
: Gets the value of a specified Flash action variable.

`FlashMediaGetRefConBounds` (page 198)
: Undocumented

`FlashMediaGetRefConID` (page 198)
: Undocumented

`FlashMediaGetSupportedSwfVersion` (page 199)
: Identifies the version of Flash that this version of QuickTime supports.

`FlashMediaIDToRefCon` (page 200)
: Undocumented

`FlashMediaSetFlashVariable` (page 200)
: Sets the specified Flash action variable to a value.

`FlashMediaSetPan` (page 201)
: Undocumented

`FlashMediaSetZoom` (page 202)
: Undocumented

`FlashMediaSetZoomRect` (page 202)
: Undocumented

`GetMovieAudioContext` (page 205)
: Returns the current audio context for a movie.

`GetMovieNaturalBoundsRect` (page 214)
: Gets a movie's natural boundary rectangle.

`GetMovieRateChangeConstraints` (page 220)
: Returns the minimum and maximum delay you can get when a movie's rate is changed.

`GetNextTrackForCompositing` (page 227)
: Determines the next track in a movie's compositing process.

`GetPrevTrackForCompositing` (page 228)
: Determines the previous track in a movie's compositing process.

`GetTimeBaseMasterOffsetTimeBase` (page 230)
: Allows an offset time base to retrieve the master time base it is attached to.

`GetTimeBaseRateChangeStatus` (page 232)
: Lets a time base client determine the time base's last rate change status.

`GetTimeBaseThreadAttachState` (page 235)
: Determines whether a given time base is attached to a thread.

`ITextAddString` (page 243)
: Undocumented

`ITextGetString` (page 244)
: Undocumented

`ITextRemoveString` (page 244)
: Undocumented

`Media3DGetCameraAngleAspect` (page 248)
: Deprecated.

`Media3DGetCameraData` (page 248)
: Deprecated.

`Media3DGetCameraRange` (page 248)
: Deprecated.

`Media3DGetCurrentGroup` (page 249)
: Deprecated.

`Media3DGetNamedObjectList` (page 249)
: Deprecated.

`Media3DGetRendererList` (page 249)
: Deprecated.

`Media3DGetViewObject` (page 250)
: Deprecated.

`Media3DRotateNamedObjectTo` (page 250)
: Deprecated.

`Media3DScaleNamedObjectTo` (page 251)
: Deprecated.

`Media3DSetCameraAngleAspect` (page 251)
: Deprecated.

`Media3DSetCameraData` (page 251)
: Deprecated.

`Media3DSetCameraRange` (page 252)
: Deprecated.

`Media3DTranslateNamedObjectTo` (page 252)
: Deprecated.

`MovieMediaGetChildDoMCActionCallback` (page 252)
: Undocumented

`MovieMediaGetChildMovieDataReference` (page 253)
: Undocumented

`MovieMediaGetCurrentMovieProperty` (page 254)
: Retrieves current properties from a media handler's movie.

`MovieMediaGetCurrentTrackProperty` (page 255)
: Retrieves the media type property from a media handler's track.

`MovieMediaGetDoMCActionCallback` (page 255)
: Gets a DoMCActionProc callback for a media.

`MovieMediaLoadChildMovieFromDataReference` (page 256)
: Undocumented

`MovieMediaSetChildMovieDataReference` (page 257)
: Undocumented

`NewMovieFromProperties` (page 260)
: Creates a new movie using movie properties.

`PutMovieIntoDataFork64` (page 267)
: Provides a 64-bit version of PutMovieIntoDataFork.

`QTAudioContextCreateForAudioDevice` (page 269)
: Creates a QTAudioContext object that encapsulates a connection to a CoreAudio output device.

`QTParseTextHREF` (page 277)
: Undocumented

`QTSoundDescriptionGetProperty` (page 280)
: Gets a particular property of a sound description.

`QTSoundDescriptionGetPropertyInfo` (page 281)
: Gets information about a particular property of a sound description.

`QTSoundDescriptionSetProperty` (page 282)
: Sets a particular property of a sound description.

`QTTextToNativeText` (page 282)
: Undocumented

`SetMovieAudioContext` (page 286)
: Targets a movie to render into an audio context.

`SetMovieVideoOutput` (page 301)
: Indicates to the ICM the video output component being used with a given movie.

`SetTimeBaseOffsetTimeBase` (page 306)
: Attaches an offset time base to another time base.

`SpriteMediaDisposeSprite` (page 315)
: Disposes of memory allocated for a sprite.

`SpriteMediaGetActionVariableAsString` (page 316)
: Undocumented

`SpriteMediaGetProperty` (page 319)
: Gets a sprite property; superseded by SpriteMediaGetSpriteProperty.

`SpriteMediaGetSpriteActionsForQTEvent` (page 320)
: Gets the sprite action atom for an event.

`SpriteMediaHitTestSprites` (page 324)
: Undocumented

`SpriteMediaNewSprite` (page 327)
: Creates a new sprite.

`SpriteMediaSetActionVariableToString` (page 328)
: Undocumented

`SpriteMediaSetProperty` (page 329)
: Sets a sprite property; superseded by SpriteMediaSetSpriteProperty.

`TextMediaDrawRaw` (page 339)
: Undocumented

`TextMediaGetTextProperty` (page 341)
: Sets properties of a text media.

`TextMediaRawIdle` (page 343)
: Undocumented

`TextMediaRawSetup` (page 344)
: Undocumented

`TextMediaSetTextProperty` (page 345)
: Sets properties of a text media.

`VideoMediaGetCodecParameter` (page 348)
: Undocumented

`VideoMediaGetStallCount` (page 349)
: Undocumented

`VideoMediaSetCodecParameter` (page 351)
: Undocumented

Functions

AbortPrePrerollMovie

Terminates the operation of PrePrerollMovie.

```
void AbortPrePrerollMovie (
   Movie m,
   OSErr err
);
```

Parameters

m

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

err

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

You normally call this function only if you have previously called `PrePrerollMovie` (page 263) asynchronously and the user quits your application.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

AddCallBackToTimeBase

Places a callback event into the list of scheduled callback events.

```
OSErr AddCallBackToTimeBase (
   QTCallBack cb
);
```

Parameters

cb

Specifies the callback event for the operation. Your clock component obtains this value from the parameters passed to `ClockCallMeWhen` (page 453).

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

If your component calls this function, the Movie Toolbox notifies it of time, rate, or stop and start changes via `ClockRateChanged` (page 458) and `ClockTimeChanged` (page 460).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

AddTime

Adds two times.

```
void AddTime (
   TimeRecord *dst,
   const TimeRecord *src
);
```

Parameters

dst

A pointer to a time structure. This time structure contains one of the operands for the addition. `AddTime` returns the result of the addition into this time structure.

src

A pointer to a time structure. The Movie Toolbox adds this value to the time or duration specified by the `dst` parameter.

Return Value

You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Discussion

You must specify the input times in time structures. The result value is formatted as a duration or a time value, the same as the format of the structure pointed to by the `dst` parameter.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

AttachTimeBaseToCurrentThread

Attaches a time base to the current thread.

```
OSErr AttachTimeBaseToCurrentThread (
   TimeBase tb
);
```

Parameters

tb
A time base.

Return Value
See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 6.4.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`Movies.h`

CallMeWhen

Schedules a callback event.

```
OSErr CallMeWhen (
   QTCallBack cb,
   QTCallBackUPP callBackProc,
   long refCon,
   long param1,
   long param2,
   long param3
);
```

Parameters

cb
The callback event for the operation. You obtain this identifier from `NewCallBack` (page 258).

callBackProc
Points to your callback function, described in `QTCallBackProc`.

refCon

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

param1

Contains scheduling information. The Movie Toolbox interprets this parameter based on the value of the `cbType` parameter to `NewCallBack` (page 258). If `cbType` is set to `callBackAtTime`, the param1 parameter contains flags (see below) indicating when to invoke your callback function for this callback event. If the `cbType` parameter is set to `callBackAtRate`, param1 contains flags (see below) indicating when to invoke your callback function for this event. Be sure to set unused flags to 0.

param2

Contains scheduling information. The Movie Toolbox interprets this parameter based on the value of the `cbType` parameter to `NewCallBack` (page 258). If `cbType` is set to `callBackAtTime`, the param2 parameter contains the time value at which your callback function is to be invoked for this event. The param1 parameter contains flags affecting when the Movie Toolbox calls your function. If `cbType` is set to `callBackAtRate`, the param2 parameter contains the rate value at which your callback function is to be invoked for this event. The param1 parameter contains flags affecting when the Movie Toolbox calls your function.

param3

The time scale in which to interpret the time value that is stored in param3 if `cbType` is set to `callBackAtTime`.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

You can call this function from your callback function.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qtbigscreen

qtbigscreen.win

Show Movie

SimpleCocoaMovie

SimpleCocoaMovieQT

Declared In

`Movies.h`

CancelCallBack

Cancels a callback event before it executes.

```
void CancelCallBack (
   QTCallBack cb
);
```

Parameters

cb

The callback event for this operation. You obtain this value from NewCallBack (page 258).

Return Value

You can access this function's error returns through GetMoviesError (page 221) and GetMoviesStickyError (page 222).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

CheckQuickTimeRegistration

Deprecated.

```
void CheckQuickTimeRegistration (
   void *registrationKey,
   long flags
);
```

Version Notes

This function is listed for historical purposes only. It may be unsupported or removed in future versions of QuickTime.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

ChooseMovieClock

Searches media handlers to find the best clock for a movie.

```
void ChooseMovieClock (
   Movie m,
   long flags
);
```

Parameters

m

The movie for this operation. Your application obtains this movie identifier from such functions as NewMovie (page 259), NewMovieFromFile (page 1398), and NewMovieFromHandle (page 1400).

flags
Currently not used; set to 0.

Discussion
This function calls `MediaGetClock` (page 1101) and finds the first media handler that has a custom clock. It then calls `SetMovieMasterClock` (page 290) to use the best clock as the movie's master timebase clock.

`ChooseMovieClock` can be used to tie the movie's master timebase to a sound clock if there is a sound track. If there is no sound track, the microseconds clock is used as the master timebase.

Version Notes
Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`Movies.h`

ClearMoviesStickyError

Clears the sticky error value.

```
void ClearMoviesStickyError (
   void
);
```

Discussion
The Movie Toolbox provides two error values to your application: the current error and the sticky error. The current error is the result code from the last Movie Toolbox function; it is updated each time your application calls a Movie Toolbox function. The Movie Toolbox saves the same result code in the sticky error value. Your application clears the sticky error value by calling `ClearMoviesStickyError`. The Movie Toolbox then places the first nonzero result code from any toolbox function used by your application into the sticky error value. The Movie Toolbox does not update the sticky error value until your application clears it again.

Special Considerations
Many Movie Toolbox functions don't return an error as a function result; you must use `GetMoviesError` to obtain the result code. Even if a function explicitly returns an error as a function result, that result is also available using `GetMoviesError`. The Movie Toolbox does not place a result code into the sticky error value until the field has been cleared. Your application is responsible for clearing the sticky error value to ensure that it does not contain a stale result code.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
BurntTextSampleCode
vrmakeobject
vrmakepano
VRMakePano Library
vrmakepano.win

Declared In
`Movies.h`

ConvertTime

Converts a time obtained from one time base into a time that is relative to another time base.

```
void ConvertTime (
   TimeRecord *theTime,
   TimeBase newBase
);
```

Parameters

theTime
: A pointer to a time structure that contains the time value to be converted. The `ConvertTime` function replaces the contents of this time structure with the time value relative to the specified time base.

newBase
: The time base for this operation. Your application obtains this time base identifier from the `NewTimeBase` (page 261) function.

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Discussion
This function includes the rate associated with each time value in the conversion; therefore, you should use this function when you want to convert time values. Both time bases must rely on the same time source, and you must specify the time to be converted in a time structure.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

ConvertTimeScale

Converts a time from one time scale into a time that is relative to another time scale.

```
void ConvertTimeScale (
   TimeRecord *theTime,
   TimeScale newScale
);
```

Parameters

theTime
: A pointer to a time structure that contains the time value to be converted. `ConvertTimeScale` replaces the contents of this time structure with the time value relative to the specified time scale.

newScale
: The time scale for this operation.

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Discussion
This function does not include the rate associated with the time value in the conversion; therefore, you should use this function when you want to convert time durations, but not when converting time values.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
AlwaysPreview
qtsndtween
qtsndtween.win
qttext.win
TimeCode Media Handlers

Declared In
`Movies.h`

ConvertTimeToClockTime

Converts a time record in a time base to clock time.

```
void ConvertTimeToClockTime (
   TimeRecord *time
);
```

Parameters

`time`
The `TimeRecord` structure to be converted. It must contain a valid time base; otherwise it remains untouched.

Discussion
The result of this call has no meaning it the time base rate is 0.

Version Notes
Introduced in QuickTime 6.4.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`Movies.h`

CreateMovieControl

Creates a movie control object to pass to the Mac OS Control Manager.

```
OSErr CreateMovieControl (
   WindowRef theWindow,
   Rect *localRect,
   Movie theMovie,
   UInt32 options,
   ControlRef *returnedControl
);
```

Parameters

theWindow

The window in which the control is placed.

localRect

A pointer to a `Rect` structure that describes in local coordinates the window in which the movie control is placed. If `NIL` is passed, the movie control is positioned at 0,0 within the window; it will have the natural dimensions of the movie plus the height of the movie controls if they are visible. If 0 height and width is passed, this parameter is interpreted as an anchor point and the top left point of the movie control will be located at this position with height and width as in the `NIL` case. For all other cases of rectangles, the movie control is centered within the rectangle by default and is sized to fit within it while maintaining the movie's aspect ratio.

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

options

Constants (see below) that determine parts of the movie control's appearance. See these constants:

`kMovieControlOptionHideController`

`kMovieControlOptionLocateTopLeft`

`kMovieControlOptionEnableEditing`

`kMovieControlOptionHandleEditingHI`

`kMovieControlOptionSetKeysEnabled`

`kMovieControlOptionManuallyIdled`

returnedControl

A handle to a `ControlRecord` struct. This defines a movie control, suitable for passing to Mac OS Control Manager functions.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`. This routine returns an error if there is a problem with one of the parameters or if an error occurred while creating the underlying movie controller or the custom control itself. If an error is returned, the value of `returnedControl` is undefined.

Discussion

The Carbon Movie Control is implemented as a custom control, which installs an event handler to handle the Carbon Events sent to controls. When a Carbon Movie Control is created for a movie, a movie controller is also created. The movie control then directs user interface events to the controller. The application can install event handlers on the Carbon Movie Control to handle such things as contextual menu clicks or to intercept events to do special processing. Control Manager calls can be made as well.

Special Considerations

The control can be deleted by calling the Mac OS function `DisposeControl`. Note that the control is automatically disposed of if the enclosing window is destroyed. Note, too, that the underlying movie controller is disposed of when the control is deleted.

Version Notes

Introduced in QuickTime 6.

Availability

Available in Mac OS X v10.2 and later.

Related Sample Code

QTCarbonShell

qtshellCEvents

qtshellCEvents.win

QuickTimeMovieControl

Declared In

`Movies.h`

DetachTimeBaseFromCurrentThread

Detaches a time base from the current thread.

```
OSErr DetachTimeBaseFromCurrentThread (
   TimeBase tb
);
```

Parameters

`tb`

A time base.

Return Value

See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 6.4.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`Movies.h`

DisposeCallBack

Disposes of a callback event.

```
void DisposeCallBack (
   QTCallBack cb
);
```

Parameters

cb

The callback event for the operation. You obtain this value from `NewCallBack` (page 258).

Return Value

You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Discussion

You should call this function when you are done with each callback event.

Special Considerations

Don't call this function at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qtbigscreen

qtbigscreen.win

Show Movie

SimpleCocoaMovie

SimpleCocoaMovieQT

Declared In

`Movies.h`

DisposeMatte

Disposes of a matte obtained from the GetTrackMatte function.

```
void DisposeMatte (
   PixMapHandle theMatte
);
```

Parameters

theMatte

`Handle` to the matte to be disposed. Your application obtains this handle from `GetTrackMatte` (page 238).

Return Value

You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Inside Mac Movie TB Code

Declared In
`Movies.h`

DisposeMovie

Frees any memory being used by a movie, including the memory used by the movie's tracks and media structures.

```
void DisposeMovie (
   Movie theMovie
);
```

Parameters

theMovie

Identifies the movie to be freed. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), or `NewMovieFromHandle` (page 1400).

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Discussion
Your application should call this function when it is done working with a movie, as shown in the following example:

```
// DisposeMovie coding example
// See "Discovering QuickTime," page 85
void CreateMyCoolMovie (void)
{
    StandardFileReply   sfr;
    Movie               movie =NIL;
    FSSpec              fss;
    short               nFileRefNum =0;
    short               nResID =movieInDataForkResID;
    StandardPutFile("\pEnter movie file name:", "\puntitled.mov", &sfr);
    if (!sfr.sfGood)
        return;
    CreateMovieFile(&sfr.sfFile,
                    FOUR_CHAR_CODE('TVOD'),
                    smCurrentScript,
                    createMovieFileDeleteCurFile |
createMovieFileDontCreateResFile,
                    &nFileRefNum,
                    &movie);
    CreateMyVideoTrack(movie);      // See "Creating a Track," below
    AddMovieResource(movie, nFileRefNum, &nResID, NIL);
    if (nFileRefNum !=0)
        CloseMovieFile(nFileRefNum);
    DisposeMovie(movie);
}
```

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
MakeEffectMovie
vrmakeobject
vrmakepano
VRMakePano Library
vrmakepano.win

Declared In
`Movies.h`

DisposeTimeBase

Disposes of a time base once you are finished with it.

```
void DisposeTimeBase (
   TimeBase tb
);
```

Parameters

tb

The time base for this operation. Your application obtains this time base identifier from `NewTimeBase` (page 261).

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qteffects
qteffects.win
qtshoweffect
qtshoweffect.win
vrscript.win

Declared In
`Movies.h`

EnterMovies

Initializes the Movie Toolbox and creates a private storage area for your application.

```
OSErr EnterMovies (
   void
);
```

Return Value

Be sure to check the value returned by this function before using any other facilities of the Movie Toolbox. See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Before calling any Movie Toolbox functions, you must use `EnterMovies` to initialize the toolbox. Your application may call `EnterMovies` multiple times. The following code sample demonstrates how your application can call the Gestalt Manager to determine whether the Movie Toolbox is installed, using the selector `gestaltQuickTime` (`'qtim'`), before calling `EnterMovies`:

```
//Using the Gestalt Manager with the Movie Toolbox
#include <GestaltEqu.h>
#include <Movies.h>
Boolean IsQuickTimeInstalled (void)
{
    short   error;
    long    result;

    error =Gestalt (gestaltQuickTime, &result);
    return (error ==noErr);
}
void main (void)
{
    Boolean qtInstalled;
    .
    .
    .
    qtInstalled =IsQuickTimeInstalled ();
}
// EnterMovies coding example
// See "Discovering QuickTime," page 242
void MyInitMovieToolbox (void)
{
    InitGraf(&qd.thePort);
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs(NIL);
    EnterMovies();
}
void main (void)
{
    MyInitMovieToolbox();
    CreateMyCoolMovie();
}
```

Special Considerations

You should initialize any other Macintosh managers your application uses before calling `EnterMovies`. You do not need to balance calls to `EnterMovies` with calls to `ExitMovies` (page 192); you need to call `ExitMovies` only if you finish with the Movie Toolbox long before your application is ready to quit.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Inside Mac Movie TB Code
MakeEffectMovie
mfc.win
MovieGWorlds
SGDataProcSample

Declared In
`Movies.h`

EnterMoviesOnThread

Indicates that the client will be using QuickTime on the current thread.

```
OSErr EnterMoviesOnThread (
   UInt32 inFlags
);
```

Parameters

`inFlags`
: Flag (see below) indicating how the executing thread will use QuickTime. Setting the thread mode is a convenience provided by this function. Pass 0 for the default options. See these constants:
  `kQTEnterMoviesFlagDontSetComponentsThreadMode`

Return Value
See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error. This function returns an appropriate operating system or QuickTime error if the operation couldn't be completed. This might occur because a second call on the thread was made that used incompatible flags (for example, the first call required a shared state but a subsequent call required a private state).

Discussion
This function is analogous to `EnterMovies`. It initializes QuickTime and prepares QuickTime for calls from its thread. Unlike `EnterMovies`, this function allows the client to indicate if its access to QuickTime requires sharing of QuickTime state with the main thread. The default is to maintain a private state.

Version Notes
Introduced in QuickTime 6.4.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
ExtractMovieAudioToAIFF
QTAudioExtractionPanel
ThreadsExportMovie
ThreadsImporter
ThreadsImportMovie

Declared In

`Movies.h`

ExecuteCallBack

Called by a clock component when it determines that it is time to execute a callback function.

```
void ExecuteCallBack (
   QTCallBack cb
);
```

Parameters

cb

Specifies the callback event for the operation. Your clock component obtains this value from the parameters passed to your `ClockCallMeWhen` (page 453) function.

Return Value

You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Discussion

Before calling the application's function, the `ExecuteCallBack` function cancels the callback event. In this manner, the callback event is prevented from executing twice in succession. It is up to the application, or the callback function itself, to reschedule the callback event.

Special Considerations

Your clock component should not release the memory associated with the callback event at this time. You should do so only with `ClockDisposeCallBack` (page 455). This is particularly important when a callback function cannot execute at interrupt time, since the Movie Toolbox schedules such functions for invocation at a later time.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

ExitMovies

Automatically called when an application quits.

```
void ExitMovies (
   void
);
```

Return Value

You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Discussion

You only need to call this function if you finish with the Movie Toolbox long before your application is ready to quit. When you call `ExitMovies`, the Movie Toolbox releases the private storage (which may be significant) that was allocated when you called `EnterMovies` (page 189). As a general rule, your application seldom uses this function; the following code illustrates an exception:

```
// ExitMovies coding example
// See "Discovering QuickTime," page 225
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                      LPSTR lpCmdLine, int nCmdShow)
{
    MSG         msg;
    HANDLE      hAccelTable;

    if  (!hPrevInstance)                   // Is there a previous instance?
        if (!(InitApplication(hInstance)))      // Register window class
            return FALSE;                       // Report failure

    if (InitializeQTML(0) !=0) {                       // Initialize QTML
        MessageBox(hwnd, "QuickTime not available",     // Notify user
                         "", MB_OK);
        return FALSE;                                   // Report failure
    }  // end if (InitializeQTML(0) !=0)

    if (EnterMovies() !=0) {                       // Initialize QuickTime
        MessageBox(hwnd, "QuickTime not available",     // Notify user
                         "", MB_OK);
        return FALSE;                                   // Report failure
    }  // end if (EnterMovies() !=0)

    if (!(InitInstance(hInstance, nCmdShow)))         // Create main window
        return FALSE;                                 // Report failure

    hAccelTable =LoadAccelerators(hInstance,     // Load accelerator table
                    MAKEINTRESOURCE(IDR_ACCELSIMPLESDI));

    //  Main message loop

    while (GetMessage(&msg, NIL, 0, 0))        // Retrieve next message
        if (!TranslateAccelerator(msg.hwnd,    // Check for kbd accelerator
                        hAccelTable, &msg)) {
            TranslateMessage(&msg);     // Convert virtual key to character
            DispatchMessage(&msg);      // Send message to window procedure
        }  // end if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))

    ExitMovies();                                   // Terminate Toolbox
    TerminateQTML();                                // Terminate QuickTime

    return msg.wParam;
}  // end WinMain
```

Special Considerations

Before calling `ExitMovies`, be sure that you have closed your connections to any components that use the Movie Toolbox, such as movie controllers, sequence grabbers, and so on.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
MakeEffectMovie

makeeffectslideshow

mfc.win

qtcontroller

qtwiredactions

Declared In
`Movies.h`

ExitMoviesOnThread

Indicates to QuickTime that the client will no longer be using QuickTime on the current thread.

```
OSErr ExitMoviesOnThread (
   void
);
```

Return Value
See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error. Returns an appropriate operating system or QuickTime error if the operation couldn't be completed. This might occur because a previous call to `EnterMoviesOnThread` was not made.

Discussion
This function should be called before exiting from a spawned thread that uses QuickTime. It undoes the setup performed by `EnterMoviesOnThread`. Each call to `EnterMoviesOnThread` should be matched with a call to this function. This function should not be called on a thread without a previous call to `EnterMoviesOnThread`.

Version Notes
Introduced in QuickTime 6.4.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
ExtractMovieAudioToAIFF

QTAudioExtractionPanel

ThreadsExportMovie

ThreadsImporter

ThreadsImportMovie

Declared In
`Movies.h`

FlashMediaDoButtonActions

Performs actions attached to a specified button.

```
ComponentResult FlashMediaDoButtonActions (
   MediaHandler mh,
   char *path,
   long buttonID,
   long transition
);
```

Parameters

mh

The Toolbox's connection to your derived Flash media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

path

Specifies the path to the button to which the action is attached.

buttonID

The ID of the button.

transition

Sends a mouse transition message to the object and whatever Flash actions are associated with that transition on the object that should be performed. The values are specific Flash transition constants.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

FlashMediaFrameLabelToMovieTime

Undocumented

```
ComponentResult FlashMediaFrameLabelToMovieTime (
   MediaHandler mh,
   Ptr theLabel,
   TimeValue *movieTime
);
```

Parameters

mh

The Toolbox's connection to your derived Flash media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

theLabel

Undocumented

movieTime

Undocumented

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

FlashMediaFrameNumberToMovieTime

Undocumented

```
ComponentResult FlashMediaFrameNumberToMovieTime (
   MediaHandler mh,
   long flashFrameNumber,
   TimeValue *movieTime
);
```

Parameters

mh
: The Toolbox's connection to your derived Flash media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

flashFrameNumber
: *Undocumented*

movieTime
: *Undocumented*

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

FlashMediaGetDisplayedFrameNumber

Undocumented

```
ComponentResult FlashMediaGetDisplayedFrameNumber (
   MediaHandler mh,
   long *flashFrameNumber
);
```

Parameters

mh

The Toolbox's connection to your derived Flash media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

flashFrameNumber

Undocumented

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

FlashMediaGetFlashVariable

Gets the value of a specified Flash action variable.

```
ComponentResult FlashMediaGetFlashVariable (
   MediaHandler mh,
   char *path,
   char *name,
   Handle *theVariableCStringOut
);
```

Parameters

mh

The Toolbox's connection to your derived Flash media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

path

Specifies the path to the Flash button to which the variable is attached.

name

Specifies the name of the Flash variable.

theVariableCStringOut

A handle to the value of the Flash variable as a C string.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

FlashMediaGetRefConBounds

Undocumented

```
ComponentResult FlashMediaGetRefConBounds (
   MediaHandler mh,
   long refCon,
   long *left,
   long *top,
   long *right,
   long *bottom
);
```

Parameters

`mh`
The Toolbox's connection to your derived Flash media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

`refCon`
Undocumented

`left`
Undocumented

`top`
Undocumented

`right`
Undocumented

`bottom`
Undocumented

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

FlashMediaGetRefConID

Undocumented

```
ComponentResult FlashMediaGetRefConID (
   MediaHandler mh,
   long refCon,
   long *refConID
);
```

Parameters

mh

The Toolbox's connection to your derived Flash media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

refCon

Undocumented

refConID

Undocumented

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

FlashMediaGetSupportedSwfVersion

Identifies the version of Flash that this version of QuickTime supports.

```
ComponentResult FlashMediaGetSupportedSwfVersion (
   MediaHandler mh,
   unsigned char *swfVersion
);
```

Parameters

mh

The Toolbox's connection to your derived Flash media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

swfVersion

The version number of the most current version of Flash that this version of QuickTime supports.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

FlashMediaIDToRefCon

Undocumented

```
ComponentResult FlashMediaIDToRefCon (
   MediaHandler mh,
   long refConID,
   long *refCon
);
```

Parameters

mh

The Toolbox's connection to your derived Flash media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

refConID

Undocumented

refCon

Undocumented

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

FlashMediaSetFlashVariable

Sets the specified Flash action variable to a value.

```
ComponentResult FlashMediaSetFlashVariable (
   MediaHandler mh,
   char *path,
   char *name,
   char *value,
   Boolean updateFocus
);
```

Parameters

mh

The Toolbox's connection to your derived Flash media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

path
: Specifies the path to the Flash button to which the variable is attached.

name
: Specifies the name of the Flash variable.

value
: Specifies the new value of the Flash variable.

updateFocus
: Pass TRUE if the focus is to be changed.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

FlashMediaSetPan

Undocumented

```
ComponentResult FlashMediaSetPan (
   MediaHandler mh,
   short xPercent,
   short yPercent
);
```

Parameters

mh
: *Undocumented*

xPercent
: *Undocumented*

yPercent
: *Undocumented*

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

FlashMediaSetZoom

Undocumented

```
ComponentResult FlashMediaSetZoom (
   MediaHandler mh,
   short factor
);
```

Parameters

mh

    Undocumented

factor

    Undocumented

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

FlashMediaSetZoomRect

Undocumented

```
ComponentResult FlashMediaSetZoomRect (
   MediaHandler mh,
   long left,
   long top,
   long right,
   long bottom
);
```

Parameters

mh

    Undocumented

left

    Undocumented

top

    Undocumented

right

    Undocumented

bottom

    Undocumented

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

GetCallBackTimeBase

Retrieves the time base of a callback event.

```
TimeBase GetCallBackTimeBase (
   QTCallBack cb
);
```

Parameters

cb

The callback event for the operation. You obtain this value from the `NewCallBack` (page 258) function.

Return Value

A pointer to a `TimeBaseRecord` structure.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

GetCallBackType

Retrieves a callback event's type.

```
short GetCallBackType (
   QTCallBack cb
);
```

Parameters

cb

The callback event for the operation. You obtain this value from `NewCallBack` (page 258).

Return Value

The callback type constant (see below). If the high-order bit (defined by `callBackAtInterrupt`) of the returned value is set to 1, the event can be invoked at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

GetFirstCallBack

Returns the first callback event associated with a specified time base.

```
QTCallBack GetFirstCallBack (
   TimeBase tb
);
```

Parameters

tb

Specifies the time base for the operation. Your component can obtain the time base reference from your `ClockSetTimeBase` (page 459) function or from the Movie Toolbox's `GetCallBackTimeBase` (page 203) function.

Return Value
A pointer to a `CallBackRecord` structure. Your software can pass this structure to other functions, such as `ClockRateChanged` (page 458).

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

GetMovieActive

Determines whether a movie is currently active.

```
Boolean GetMovieActive (
   Movie theMovie
);
```

Parameters

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

Return Value
TRUE if the movie is currently active, FALSE otherwise.

Special Considerations

The Movie Toolbox services only active movies.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

GetMovieActiveSegment

Determines what portion of a movie is currently active for playing.

```
void GetMovieActiveSegment (
   Movie theMovie,
   TimeValue *startTime,
   TimeValue *duration
);
```

Parameters

theMovie
The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

startTime
A pointer to a time value. `GetMovieActiveSegment` places the starting time of the active segment into the field referred to by this parameter. If the returned time value is set to -1, the entire movie is active. In this case, the Movie Toolbox does not return any duration information.

duration
A pointer to a time value. `GetMovieActiveSegment` places the duration of the active movie segment into the field referred to by this parameter. If the entire movie is active, the `startTime` parameter is set to -1 and this parameter does not return any duration information.

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

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.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`Movies.h`

GetMovieBoundsRgn

Determines a movie's boundary region.

```
RgnHandle GetMovieBoundsRgn (
   Movie theMovie
);
```

Parameters

theMovie
: The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

Return Value
A handle to a `MacRegion` structure that the function allocates. If the movie does not have a spatial representation at the current time, the function returns an empty region. If the function could not satisfy the request, it sets the returned handle to `NIL`.

Discussion
The Movie Toolbox derives the boundary region only from enabled tracks, and only from those tracks that are used in the current display mode (that is, movie or preview). The boundary region is valid for the current movie time.

Special Considerations

Your application must dispose of the returned region when it is done with it.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

GetMovieBox

Returns a movie's boundary rectangle, which is a rectangle that encompasses all of the movie's enabled tracks.

```
void GetMovieBox (
   Movie theMovie,
   Rect *boxRect
);
```

Parameters

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as NewMovie (page 259), NewMovieFromFile (page 1398), and NewMovieFromHandle (page 1400).

boxRect

A pointer to a rectangle. GetMovieBox returns the coordinates of the movie's boundary rectangle into the structure referred to by this parameter.

Return Value

You can access this function's error returns through GetMoviesError (page 221) and GetMoviesStickyError (page 222).

Discussion

The movie box is in the coordinate system of the movie's graphics world and defines the movie's boundaries over the entire duration of the movie. The movie's boundary rectangle defines the size and shape of the movie before the Movie Toolbox applies the display clipping region. The following code sample illustrates the use of GetMovieBox:

```
// GetMovieBox coding example
// See "Discovering QuickTime," page 218
void main (void)
{
    WindowRef       pMacWnd;
    Rect            rectWnd;
    Rect            rectMovie;
    Movie           movie;
    Boolean         bDone =FALSE;
    OSErr           nErr;
    EventRecord     er;
    WindowRef       pWhichWnd;
    short           nPart;
    InitGraf(&qd.thePort);
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs(NIL);
    nErr =EnterMovies();
    if (nErr !=noErr)
        return;

    SetRect(&rectWnd, 100, 100, 200, 200);
    pMacWnd =NewCWindow(NIL, &rectWnd, "\pMovie", FALSE,
                        noGrowDocProc, (WindowRef)-1, TRUE, 0);
    SetPort(pMacWnd);
    movie =GetMovie();
    if (movie ==NIL)
```

```
        return;

    GetMovieBox(movie, &rectMovie);
    OffsetRect(&rectMovie, -rectMovie.left, -rectMovie.top);
    SetMovieBox(movie, &rectMovie);

    SizeWindow(pMacWnd, rectMovie.right, rectMovie.bottom, TRUE);
    ShowWindow(pMacWnd);
    SetMovieGWorld(movie, (CGrafPtr)pMacWnd, NIL);

    StartMovie(movie);

    . . .

    DisposeMovie(movie);
    DisposeWindow(pMacWnd);
}
```

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
MakeEffectMovie

qteffects

vrmovies

vrscript

vrscript.win

Declared In
`Movies.h`

GetMovieClipRgn

Determines a movie's clipping region.

```
RgnHandle GetMovieClipRgn (
   Movie theMovie
);
```

Parameters

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

Return Value
A handle to a `MacRegion` structure, which the function allocates, that represents the clipping region. If the function could not satisfy your request or if there is no clipping region defined for the movie, `GetMovieClipRgn` sets the returned handle to `NIL`.

Discussion
The clipping region is saved with the movie when your application saves the movie.

Special Considerations

Your application must dispose of this region when it is done with it.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

GetMovieCreationTime

Returns the movie's creation date and time information.

```
unsigned long GetMovieCreationTime (
   Movie theMovie
);
```

Parameters

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

Return Value

The movie's creation date and time.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

GetMovieDisplayBoundsRgn

Determines a movie's display boundary region.

```
RgnHandle GetMovieDisplayBoundsRgn (
   Movie theMovie
);
```

Parameters

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

Return Value

A handle to a `MacRegion` structure that the function allocates. If the movie does not have a spatial representation at the current time, the function returns an empty region. If the function could not satisfy the request, it sets the returned handle to `NIL`.

Discussion

The display boundary region encloses all of a movie's enabled tracks after the track matrix, track clip, movie matrix, and movie clip have been applied to them. This region is in the display coordinate system of the movie's graphics world. The Movie Toolbox derives the display boundary region only from enabled tracks, and only from those tracks that are used in the current display mode (that is, movie, poster, or preview). The display boundary region is valid for the current movie time.

Special Considerations

Your application must dispose of the returned handle when it is done with it.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Inside Mac Movie TB Code

Declared In

`Movies.h`

GetMovieDisplayClipRgn

Determines a movie's current display clipping region.

```
RgnHandle GetMovieDisplayClipRgn (
   Movie theMovie
);
```

Parameters

`theMovie`

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

Return Value

A handle to a `MacRegion` structure that the function allocates. If the movie does not have a spatial representation at the current time, the function returns an empty region. If the function could not satisfy the request, it sets the returned handle to `NIL`.

Special Considerations

Your application must dispose of the returned handle when it is done with it. Note that the display clipping region is not saved with the movie.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

GetMovieDuration

Returns the duration of a movie.

```
TimeValue GetMovieDuration (
   Movie theMovie
);
```

Parameters

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

Return Value

The duration of the designated movie.

Discussion

This function returns a time value, expressed in the movie's time scale, that is calculated to be the maximum durations of all the tracks in the movie. The following code sample illustrates its use:

```
// GetMovieDuration coding example
// See "Discovering QuickTime," page 363
Movie           movie1;
TimeValue       lOldDuration;
Movie           movie2;
long            lIndex, lOrigTrackCount, lReferenceIndex;
Track           track, trackSprite;
// get the first track in original movie and position at the start
trackSprite =GetMovieIndTrack(movie1, 1);
SetMovieSelection(movie1, 0, 0);
// remove all tracks except video in modifier movie
for (lIndex =1; lIndex <=GetMovieTrackCount(movie2); lIndex++) {
    Track       track =GetMovieIndTrack(movie2, lIndex);
    OSType      dwType;
    GetMediaHandlerDescription(GetTrackMedia(track),
                                &dwType, NIL, NIL);
    if (dwType !=VideoMediaType) {
        DisposeMovieTrack(track);
        lIndex--;
    }
}
// add the modifier track to original movie
lOldDuration =GetMovieDuration(movie1);
AddMovieSelection(movie1, movie2);
DisposeMovie(movie2);
// truncate the movie to the length of the original track
DeleteMovieSegment(movie1, lOldDuration,
                    GetMovieDuration(movie1) - lOldDuration);
// associate the modifier track with the original sprite track
track =GetMovieIndTrack(movie1, lOrigTrackCount + 1);
AddTrackReference(trackSprite, track, kTrackModifierReference,
                    &lReferenceIndex);
```

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CompressMovies

MovieGWorlds

QT Internals

qtstreamsplicer

qtstreamsplicer.win

Declared In

`Movies.h`

GetMovieGWorld

Returns a movie's graphics world.

```
void GetMovieGWorld (
   Movie theMovie,
   CGrafPtr *port,
   GDHandle *gdh
);
```

Parameters

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

port

A pointer to a field that is to receive a pointer to a `CGrafPort` structure. Set this parameter to `NIL` if you don't want this information.

gdh

A pointer to a field that is to receive a handle to a `GDevice` structure. Set this parameter to `NIL` if you don't want this information.

Return Value

You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

BurntTextSampleCode

Inside Mac Movie TB Code

MovieGWorlds

vrscript

vrscript.win

Declared In

`Movies.h`

GetMovieMatrix

Retrieves a movie's transformation matrix.

```
void GetMovieMatrix (
   Movie theMovie,
   MatrixRecord *matrix
);
```

Parameters

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

matrix

A pointer to a `MatrixRecord` structure, where `GetMovieMatrix` returns the movie's matrix.

Return Value

You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

BurntTextSampleCode

QTCarbonShell

qtskins.win

VideoFrameToGWorld

vrmovies

Declared In

`Movies.h`

GetMovieModificationTime

Returns a movie's modification date and time.

```
unsigned long GetMovieModificationTime (
   Movie theMovie
);
```

Parameters

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

Return Value

The movie's modification date and time.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

GetMovieNaturalBoundsRect

Gets a movie's natural boundary rectangle.

```
void GetMovieNaturalBoundsRect (
   Movie theMovie,
   Rect *naturalBounds
);
```

Parameters

theMovie
A movie identifier. Your application obtains this identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

naturalBounds
A pointer to a `Rect` structure that represents the movie's bounding rectangle.

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CarbonQTGraphicImport
QTCarbonShell
SimpleVideoOut
ThreadsExportMovie
ThreadsImportMovie

Declared In
`Movies.h`

GetMoviePict

Creates a QuickDraw picture from a specified movie at a specified time.

```
PicHandle GetMoviePict (
   Movie theMovie,
   TimeValue time
);
```

Parameters

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

time

The movie time from which the image is to be taken.

Return Value

A handle to a `Picture` structure. If the function could not create the picture, the returned handle is set to `NIL`.

Discussion

This function uses only those movie tracks that are currently enabled and would therefore be used in playback. Your application may call this function even if the movie is inactive.

Special Considerations

Your application must dispose of this picture handle.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

AlwaysPreview

DigitizerShell

DragAndDrop Shell

MovieGWorlds

VelEng Wavelet

Declared In

`Movies.h`

GetMoviePosterPict

Creates a QuickDraw picture that contains a movie's poster.

```
PicHandle GetMoviePosterPict (
   Movie theMovie
);
```

Parameters

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

Return Value

A handle to a `Picture` structure. If the function could not create the picture, the returned handle is set to `NIL`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

bMoviePaletteCocoa

CompressMovies

MovieGWorlds

qtcompress.win

qtinfo

Declared In

`Movies.h`

GetMoviePosterTime

Returns the poster's time in a movie.

```
TimeValue GetMoviePosterTime (
   Movie theMovie
);
```

Parameters

`theMovie`

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

Return Value

The time in the movie from which its poster is taken.

Discussion

Since a movie poster has no duration, it is defined by a point in time within the movie. The time value returned by `GetMoviePosterTime` is in the time coordinate system of the movie and represents the starting time for the movie frame that contains the poster image.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

MakeEffectMovie

qteffects

qteffects.win

samplemakeeffectmovie

samplemakeeffectmovie.win

Declared In
`Movies.h`

GetMoviePreferredRate

Returns a movie's default playback rate.

```
Fixed GetMoviePreferredRate (
   Movie theMovie
);
```

Parameters

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

Return Value
The movie's default playback rate.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
MovieGWorlds

qtbigscreen

qtbigscreen.win

vrscript

vrscript.win

Declared In
`Movies.h`

GetMoviePreferredVolume

Returns a movie's preferred volume setting.

```
short GetMoviePreferredVolume (
   Movie theMovie
);
```

Parameters

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

Return Value
The movie's preferred volume setting.

Discussion
A movie's tracks have their own volume settings. A track's volume is scaled by the movie's volume to produce the track's final volume. On Macintosh computers, the movie's volume is further scaled by the sound volume that the user controls from the Sound control panel.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
MakeEffectMovie

qteffects.win

qtgraphics.win

qtwiredactions

vrbackbuffer.win

Declared In
`Movies.h`

GetMoviePreviewMode

Determines whether a movie is in preview mode.

```
Boolean GetMoviePreviewMode (
   Movie theMovie
);
```

Parameters

theMovie
The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

Return Value
TRUE if the movie is in preview mode; FALSE if the movie is in normal playback mode.

Discussion
If a movie is in preview mode, only the movie's preview can be displayed.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

GetMoviePreviewTime

Returns the starting time and duration of the movie's preview.

```
void GetMoviePreviewTime (
   Movie theMovie,
   TimeValue *previewTime,
   TimeValue *previewDuration
);
```

Parameters

theMovie
The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

previewTime
A pointer to a time value. The Movie Toolbox places the preview's starting time into the field referred to by this parameter. If the movie does not have a preview, the Movie Toolbox sets this returned value to 0.

previewDuration
A pointer to a time value. The Movie Toolbox places the preview's duration into the field referred to by this parameter. If the movie does not have a preview, the Movie Toolbox sets this returned value to 0.

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qtinfo
qtinfo.win

Declared In
`Movies.h`

GetMovieRate

Returns a movie's playback rate.

```
Fixed GetMovieRate (
   Movie theMovie
);
```

Parameters

theMovie
The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

Return Value
The rate at which the movie is currently playing, expressed as a 32-bit fixed-point number. Positive integers indicate forward rates and negative integers indicate reverse rates. A value of 1 indicates normal speed, a value of 2 indicates double speed, -2 means the movie is playing backward at double speed, and so on. A value of 0 means the movie is paused or stopped.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QT Internals

QTCarbonShell

vrscript

vrscript.win

Declared In
`Movies.h`

GetMovieRateChangeConstraints

Returns the minimum and maximum delay you can get when a movie's rate is changed.

```
OSErr GetMovieRateChangeConstraints (
   Movie theMovie,
   TimeRecord *minimumDelay,
   TimeRecord *maximumDelay
);
```

Parameters

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

minimumDelay
: A pointer to a `TimeRecord` structure. The function updates this structure to contain the minimum delay when a rate change happens.

maximumDelay
: A pointer to a `TimeRecord` structure. The function updates this structure to contain the maximum delay when a rate change happens.

Discussion
If the time base master clock of the movie is changed, this function must be called again to reflect the current constraints.

Version Notes
Introduced in QuickTime 6.4.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`Movies.h`

GetMovieSelection

Returns information about a movie's current selection.

```
void GetMovieSelection (
   Movie theMovie,
   TimeValue *selectionTime,
   TimeValue *selectionDuration
);
```

Parameters

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

selectionTime

A pointer to a time value. The `GetMovieSelection` function places the starting time of the current selection into the field referred to by this parameter. Set this parameter to `NIL` if you don't want this information.

selectionDuration

A pointer to a time value. The `GetMovieSelection` function places the duration of the current selection into the field referred to by this parameter. Set this parameter to `NIL` if you don't want this information.

Return Value

You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qtinfo

qtinfo.win

Declared In

`Movies.h`

GetMoviesError

Returns the contents of the current error value and resets the current error value to 0.

```
OSErr GetMoviesError (
   void
);
```

Return Value

See `Error Codes`. Returns `noErr` if there is no error in the current error value.

Discussion

The Movie Toolbox provides two error values to your application: the current error and the sticky error. The current error is the result code from the last Movie Toolbox function; it is updated each time your application calls a Movie Toolbox function. The following code sample shows a typical use:

```
// GetMoviesError coding example
// See "Discovering QuickTime," page 256
```

```
OSErr QTUtils_SaveMovie (Movie theMovie)
{
    StandardFileReply   mySFReply;
    StringPtr   myPrompt =QTUtils_ConvertCToPascalString(kSavePrompt);
    StringPtr   myFileName =
                    QTUtils_ConvertCToPascalString(kSaveMovieFileName);
    OSErr       myErr =noErr;
    if (theMovie ==NIL)
        return(invalidMovie);
    StandardPutFile(myPrompt, myFileName, &mySFReply);
    if (mySFReply.sfGood) {
        FlattenMovieData(   theMovie,
                            flattenAddMovieToDataFork,
                            &mySFReply.sfFile,
                            FOUR_CHAR_CODE('TVOD'),
                            smSystemScript,
                            createMovieFileDeleteCurFile);
        myErr =GetMoviesError();
    }
    free(myPrompt);
    free(myFileName);
    return(myErr);
}
```

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CompressMovies
DigitizerShell
DragAndDrop Shell
MovieGWorlds
QT Internals

Declared In
`Movies.h`

GetMoviesStickyError

Returns the contents of the sticky error value.

```
OSErr GetMoviesStickyError (
   void
);
```

Return Value
See `Error Codes`. Returns `noErr` if there is no error in the sticky error value.

Discussion
The sticky error value contains the first nonzero result code from any Movie Toolbox function that you called after having cleared the sticky error with `ClearMoviesStickyError` (page 182).

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
BurntTextSampleCode
vrmakeobject
vrmakepano
VRMakePano Library
vrmakepano.win

Declared In
`Movies.h`

GetMovieTime

Returns a movie's current time both as a time value and in a time structure.

```
TimeValue GetMovieTime (
   Movie theMovie,
   TimeRecord *currentTime
);
```

Parameters

theMovie
The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

currentTime
A pointer to a `TimeRecord` structure. The function updates this time structure to contain the movie's current time. If you don't want this information, set this parameter to `NIL`.

Return Value
The time value of the current time.

Discussion
This function returns the movie's current time value in two formats: as a time value and in a time structure.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
DigitizerShell
DragAndDrop Shell
MovieBrowser
MovieGWorlds
QT Internals

Declared In
`Movies.h`

GetMovieTimeBase

Returns a movie's time base.

```
TimeBase GetMovieTimeBase (
   Movie theMovie
);
```

Parameters

theMovie
The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

Return Value
The movie's `TimeBaseRecord` structure.

Special Considerations

The Movie Toolbox disposes of a movie's time base when you dispose of the movie.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
LiveVideoMixer2

qtinfo

vrmovies

vrscript

vrscript.win

Declared In
`Movies.h`

GetMovieTimeScale

Returns the time scale of a movie.

```
TimeScale GetMovieTimeScale (
   Movie theMovie
);
```

Parameters

theMovie
The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

Return Value
A long integer that contains the movie's time scale.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
MakeEffectMovie
qteffects
qteffects.win
qtstreamsplicer
qtstreamsplicer.win

Declared In
`Movies.h`

GetMovieUserData

Obtains access to a movie's user data list.

```
UserData GetMovieUserData (
   Movie theMovie
);
```

Parameters

theMovie
The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

Return Value
The `UserDataRecord` structure for the movie. If the function could not locate the movie's user data, it sets this return value to `NIL`.

Discussion
This function returns a reference to the movie's user data list, which is valid until you dispose of the movie. When you save the movie, the Movie Toolbox saves the user data as well.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
MakeEffectMovie
qtactiontargets
qtactiontargets.win
qtwiredactions
qtwiredactions.win

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 (page 302)), noErr is returned and visualContextOut receives NULL.

Availability

Available in Mac OS X v10.3 and later.

Declared In

Movies.h

GetMovieVolume

Returns a movie's current volume setting.

```
short GetMovieVolume (
   Movie theMovie
);
```

Parameters

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as NewMovie (page 259), NewMovieFromFile (page 1398), and NewMovieFromHandle (page 1400).

Return Value

The current volume setting for the movie.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

MakeEffectMovie

qteffects.win

qtgraphics.win

vrscript

vrscript.win

Declared In

`Movies.h`

GetNextCallBack

Returns the next callback event associated with a specified time base.

```
QTCallBack GetNextCallBack (
   QTCallBack cb
);
```

Parameters

`cb`

Specifies the starting callback event for the operation. Your clock component obtains this value from the `GetFirstCallBack` (page 204) function or from previous calls to the `GetNextCallBack` function.

Return Value

A pointer to a `CallBackRecord` structure. Your software can pass this structure to other functions, such as `ClockRateChanged` (page 458).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

GetNextTrackForCompositing

Determines the next track in a movie's compositing process.

```
Track GetNextTrackForCompositing (
   Movie theMovie,
   Track theTrack
);
```

Parameters

`theMovie`

A movie identifier. Your application obtains this identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

`theTrack`

The identifier of the track from which to start.

Return Value

The returned identifier of the next track to be composited.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

GetPrevTrackForCompositing

Determines the previous track in a movie's compositing process.

```
Track GetPrevTrackForCompositing (
   Movie theMovie,
   Track theTrack
);
```

Parameters

theMovie
: A movie identifier. Your application obtains this identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

theTrack
: The identifier of the track from which to start.

Return Value
The returned identifier of the previous track in the compositing process.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

GetTimeBaseEffectiveRate

Returns the effective rate at which the specified time base is moving relative to its master clock.

```
Fixed GetTimeBaseEffectiveRate (
   TimeBase tb
);
```

Parameters

tb
: The time base for this operation. Your application obtains this time base identifier from the `NewTimeBase` (page 261) function.

Return Value
The effective rate at which the time base specified by `tb` is moving relative to its master clock.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

GetTimeBaseFlags

Obtains the control flags of a time base.

```
long GetTimeBaseFlags (
   TimeBase tb
);
```

Parameters

tb

The time base for this operation. Your application obtains this time base identifier from `NewTimeBase` (page 261).

Return Value
Control flags (see below). Unused flags are set to 0.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
MakeEffectMovie

vrmovies

vrmovies.win

vrscript

vrscript.win

Declared In
`Movies.h`

GetTimeBaseMasterClock

Determines the clock component that is assigned to a time base.

```
ComponentInstance GetTimeBaseMasterClock (
   TimeBase tb
);
```

Parameters

tb

The time base for this operation. Your application obtains this time base identifier from the `NewTimeBase` (page 261) function.

Return Value
A reference to a component instance. If a clock component is not assigned to the time base, the returned reference is `NIL`. In this case, the time base relies on another time base for its time source. Use `GetTimeBaseMasterTimeBase` (page 231) to obtain the time base reference to that master time base.

Discussion
This function returns a reference to a component instance of the clock component that provides a time source to the specified time base. Every time base derives its time from either a clock component or from another time base. If a time base derives its time from a clock component, use this function to obtain the component instance of the clock component.

Special Considerations

The Component Manager allows a single component to serve multiple client applications at the same time. Each client application has a unique connection to the component, identified by a component instance. Don't close this connection; the time base is using it to maintain its time source.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
BrideOfMungGrab

QTMusicToo

Declared In
`Movies.h`

GetTimeBaseMasterOffsetTimeBase

Allows an offset time base to retrieve the master time base it is attached to.

```
TimeBase GetTimeBaseMasterOffsetTimeBase (
   TimeBase tb
);
```

Parameters

tb

An offset time base.

Return Value
The master time base for the offset time base passed in `tb`. Returns `NIL` if `tb` does not contain an offset time base.

Version Notes
Introduced in QuickTime 6.4.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`Movies.h`

GetTimeBaseMasterTimeBase

Determines the master time base that is assigned to a time base.

```
TimeBase GetTimeBaseMasterTimeBase (
   TimeBase tb
);
```

Parameters

tb

The time base for this operation. Your application obtains this time base identifier from `NewTimeBase` (page 261).

Return Value

A time base. If a master time base is not assigned to the time base, this function sets the returned reference to `NIL`. In this case, the time base relies on a clock component for its time source. Use `GetTimeBaseMasterClock` (page 229) to obtain the component instance reference to that clock component.

Discussion

This function returns a reference to the master time base that provides a time source to this time base. A time base derives its time from either a clock component or from another time base. If a time base derives its time from another time base, use this function to obtain the identifier for that master time base.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

GetTimeBaseRate

Retrieves the rate of a time base.

```
Fixed GetTimeBaseRate (
   TimeBase tb
);
```

Parameters

tb

The time base for this operation. Your application obtains this time base identifier from the `NewTimeBase` (page 261) function.

Return Value

The time base's rate. This rate value may be nonzero even if the time base has stopped, because it has reached its stop time. Rates may be set to negative values, which cause time to move backward for the time base.

Discussion

This function returns the current rate of the time base as a fixed-point number.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
SoftVDigX

Declared In
`Movies.h`

GetTimeBaseRateChangeStatus

Lets a time base client determine the time base's last rate change status.

```
OSErr GetTimeBaseRateChangeStatus (
   TimeBase tb,
   TimeScale scale,
   Fixed *ratedChangedTo,
   TimeBaseStatus *flags,
   TimeRecord *rateChangeTimeBaseTime,
   TimeRecord *rateChangeClockTime,
   TimeRecord *currentClockTime
);
```

Parameters

tb
A pointer to a `TimeBaseRecord` structure.

scale
The scale to use for the returned time values. Pass 0 to retrieve the time in the preferred time scale of the time base.

rateChangedTo
The rate value changed to. Clients may pass `NIL` if they do not want to receive this information.

flags
A pointer to a flag (see below) that will be returned when the clock is waiting for a future time to start moving while its rate is nonzero. When set, the unpinned time will return a negative value telling how far you are from the real start time. Clients may pass `NIL` if they do not want to receive this information. `rateChangeTimeBaseTime` The time base time when the rate changed. Clients may pass `NIL` if they do not want to receive this information. `rateChangeClockTime` The clock time when the rate changed. Clients may pass `NIL` if they do not want to receive this information. `currentClockTime` The current clock time value. Clients may pass `NIL` if they do not want to receive this information. `timeBaseRateChanging` The clock is waiting for a future time to start moving while its rate is nonzero. When set, the unpinned time will return a negative value telling how far you are from the real start time. See these constants:
`timeBaseRateChanging`

rateChangeTimeBaseTime
The time base time when the rate changed. Clients may pass `NIL` if they do not want to receive this information.

rateChangeClockTime
The clock time when the rate changed. Clients may pass `NIL` if they do not want to receive this information.

currentClockTime

The current clock time value. Clients may pass `NIL` if they do not want to receive this information.

Discussion

When the flag `timeBaseRateChanging` is returned, the amount of time left before the time base ticks is equal to (`rateChangeClockTime` - `currentClockTime`).

Version Notes

Introduced in QuickTime 6.4.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`Movies.h`

GetTimeBaseStartTime

Determines the start time of a time base.

```
TimeValue GetTimeBaseStartTime (
   TimeBase tb,
   TimeScale s,
   TimeRecord *tr
);
```

Parameters

tb

The time base for this operation. Your application obtains this time base identifier from the `NewTimeBase` (page 261) function.

s

The time scale in which to return the start time.

tr

A pointer to a time structure that is to receive the start time. This is an optional parameter. If you don't want the time value represented in a time structure, set this parameter to `NIL`.

Return Value

The time base's start time.

Discussion

This function returns a time value that contains the start time for the specified time base in the specified time scale. The function returns this value even if you specify a time structure with the `tr` parameter.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

GetTimeBaseStatus

Determines when the current time of a time base would fall outside of the range of values specified by the time base's start and stop times.

```
long GetTimeBaseStatus (
   TimeBase tb,
   TimeRecord *unpinnedTime
);
```

Parameters

tb

The time base for this operation. Your application obtains this time base identifier from the `NewTimeBase` (page 261) function.

unpinnedTime

A pointer to a time structure that is to receive the current time of the time base. Note that this time value may be outside the range of values specified by the start and stop times of the time base.

Return Value

Status flags (see below).

Discussion

The status information returned by this function allows you to determine when the current time of a time base would fall outside of the range of values specified by the start and stop times of the time base. This can happen when a time base relies on a master time base or when its time has reached the stop time.

Special Considerations

This function returns no error codes.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

GetTimeBaseStopTime

Determines the stop time of a time base.

```
TimeValue GetTimeBaseStopTime (
   TimeBase tb,
   TimeScale s,
   TimeRecord *tr
);
```

Parameters

tb

The time base for this operation. Your application obtains this time base identifier from the `NewTimeBase` (page 261) function.

s

The time scale in which to return the stop time.

tr

A pointer to a time structure that is to receive the stop time. This is an optional parameter. If you don't want the time value represented in a time structure, set this parameter to `NIL`.

Return Value

The time base's stop time.

Discussion

This function returns a time value that contains the stop time for the specified time base in the specified time scale. The function returns this value even if you specify a time structure with the `tr` parameter.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

GetTimeBaseThreadAttachState

Determines whether a given time base is attached to a thread.

```
OSErr GetTimeBaseThreadAttachState (
   TimeBase inTimeBase,
   Boolean *outAttachedToCurrentThread,
   Boolean *outAttachedToAnyThread
);
```

Parameters

inTimeBase

A time base.

outAttachedToCurrentThread

A pointer to a Boolean that on exit is TRUE if the time base is attached to the current thread, FALSE otherwise.

outAttachedToAnyThread

A pointer to a Boolean that on exit is TRUE if the time base is attached to any thread, FALSE otherwise.

Return Value

See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 6.4.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`Movies.h`

GetTimeBaseTime

Obtains the current time value from a time base.

```
TimeValue GetTimeBaseTime (
   TimeBase tb,
   TimeScale s,
   TimeRecord *tr
);
```

Parameters

tb

The time base for this operation. Your application obtains this time base identifier from the `NewTimeBase` (page 261) function.

s

The time scale in which to return the current time value. Set this parameter to 0 to retrieve the time in the preferred time scale of the time base.

tr

A pointer to a time structure that is to receive the current time value. This is an optional parameter. If you don't want the time value represented in a time structure, set this parameter to `NIL`.

Return Value

The time base's current time.

Discussion

This function returns a time value that contains the current time from the specified time base in the specified time scale. The function returns this value even if you specify a time structure with the `tr` parameter.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

AlwaysPreview

BrideOfMungGrab

QTMusicToo

SoftVDigX

Declared In

`Movies.h`

GetTrackBoundsRgn

Lets the media limit the size of a track boundary rectangle.

```
RgnHandle GetTrackBoundsRgn (
   Track theTrack
);
```

Parameters

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

Return Value

A handle to the region limited by the media.

Discussion

Because the media limits the size of the track boundary rectangle, the region returned by `GetTrackBoundsRgn` may not be rectangular and may be smaller than the track boundary region.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

GetTrackClipRgn

Determines the clipping region of a track.

```
RgnHandle GetTrackClipRgn (
   Track theTrack
);
```

Parameters

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

Return Value

A handle to the track's clipping region.

Discussion

This function allocates the region and returns a handle to the region. Your application must dispose of this region when you are done with it. If the function could not satisfy your request or if there is no clipping region defined for the track, it sets the returned handle to `NIL`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

GetTrackDisplayBoundsRgn

Determines the region a track occupies in a movie's graphics world.

```
RgnHandle GetTrackDisplayBoundsRgn (
   Track theTrack
);
```

Parameters

theTrack

The track for this operation. Your application obtains this track identifier from such functions as NewMovieTrack (page 1628) and GetMovieTrack (page 1601).

Return Value

A handle to the region the specified track occupies in a movie's graphics world.

Discussion

This function allocates the region and returns a handle to the region. If the track does not have a spatial representation at the current movie time, the function returns an empty region. If the function could not satisfy your request, it sets the returned handle to NIL.

Special Considerations

Your application must dispose of the returned region when you are done with it.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

BurntTextSampleCode

Declared In

Movies.h

GetTrackMatte

Retrieves a copy of a track's matte.

```
PixMapHandle GetTrackMatte (
   Track theTrack
);
```

Parameters

theTrack

The track for this operation. Your application obtains this track identifier from such functions as NewMovieTrack (page 1628) and GetMovieTrack (page 1601).

Return Value

A handle to a PixMap structure that represents the specified track's matte. If the function could not satisfy your request, it sets the returned handle to NIL.

Discussion

The matte defines which of the track's pixels are displayed in a movie, and it is valid for the entire duration of the movie.

Special Considerations

You should use DisposeMatte (page 187) to dispose of the matte when you are finished with it.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Inside Mac Movie TB Code

Declared In
`Movies.h`

GetTrackMovieBoundsRgn

Determines the region the track occupies in a movie's boundary region.

```
RgnHandle GetTrackMovieBoundsRgn (
   Track theTrack
);
```

Parameters

theTrack
The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

Return Value
A handle to the region the specified track occupies in its movie's boundary region. If the track does not have a spatial representation at the current movie time, the function returns an empty region. If the function could not satisfy your request, it sets the returned handle to `NIL`.

Discussion
This function determines the region by applying the track's clipping region and matrix. This region is valid only for the current movie time. The function allocates the region and returns a handle to it.

Special Considerations

Your application must dispose of the returned region when you are done with it.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

GetTrackPict

Creates a QuickDraw picture from a specified track at a specified time.

```
PicHandle GetTrackPict (
   Track theTrack,
   TimeValue time
);
```

Parameters

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

time

The time at which the image is taken.

Return Value

A handle to the specified picture. If the function could not create the picture, the returned handle is set to `NIL`.

Special Considerations

Your application must dispose of the returned picture handle.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

MakeEffectMovie

qteffects

qteffects.win

samplemakeeffectmovie

samplemakeeffectmovie.win

Declared In

`Movies.h`

GoToBeginningOfMovie

Repositions a movie to play from its start.

```
void GoToBeginningOfMovie (
   Movie theMovie
);
```

Parameters

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

Discussion

If the movie is in preview mode, the function goes to the start of the preview segment of the movie. In all other cases, this function goes to the start of the movie, where the movie time value is 0. You can access error returns from this function through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Graphic Import-Export
MovieGWorlds
OpenGLMovieQT
vrscript
vrscript.win

Declared In
`Movies.h`

GoToEndOfMovie

Repositions a movie to play from its end.

```
void GoToEndOfMovie (
   Movie theMovie
);
```

Parameters

theMovie
The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

InvalidateMovieRegion

Invalidates a small area of a movie.

```
OSErr InvalidateMovieRegion (
   Movie theMovie,
   RgnHandle invalidRgn
);
```

Parameters

theMovie
: The movie whose area you wish to invalidate. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

invalidRgn
: A region indicating the area of the movie to invalidate. If necessary, QuickTime will make a copy of this region. To invalidate the entire movie area, pass `NIL` for this parameter.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
Use this function instead of `UpdateMovie` (page 347) to invalidate a small area of a movie. It marks all areas of the movie that intersect the `invalidRgn` parameter. The next time you call `MoviesTask` (page 257), the Movie Toolbox redraws the marked areas. This provides a way to invalidate a portion of the movie's area instead of its entire area, as does `UpdateMovie`. This allows for higher performance update handling when a movie has many tracks or covers a large area. For handling of update events, applications should continue to use `UpdateMovie`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

IsMovieDone

Determines if a particular movie has completely finished playing.

```
Boolean IsMovieDone (
   Movie theMovie
);
```

Parameters

theMovie
: The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

Return Value
Returns TRUE if the specified movie has finished playing, otherwise returns FALSE.

Discussion

A movie with a positive rate (playing forward) is considered done when its movie time reaches the movie end time. Conversely, a movie with a negative rate (playing backward) is considered done when its movie time reaches the movie start time. If your application has changed the movie's active segment, the status returned by this function is relative to the active segment, rather than to the entire movie. You can use `SetMovieActiveSegment` (page 285) to change a movie's active segment.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Graphic Import-Export

MovieGWorlds

SimpleCocoaMovie

vrscript

vrscript.win

Declared In

`Movies.h`

ITextAddString

Undocumented

```
OSErr ITextAddString (
   QTAtomContainer container,
   QTAtom parentAtom,
   RegionCode theRegionCode,
   ConstStr255Param theString
);
```

Parameters

`container`
: *Undocumented*

`parentAtom`
: *Undocumented*

`theRegionCode`
: A 16-bit signed integer containing an international region code; see `Localization Codes`.

`theString`
: The string to add.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

ITextGetString

Undocumented

```
OSErr ITextGetString (
   QTAtomContainer container,
   QTAtom parentAtom,
   RegionCode requestedRegion,
   RegionCode *foundRegion,
   StringPtr theString
);
```

Parameters

`container`
    Undocumented

`parentAtom`
    Undocumented

`requestedRegion`
    Undocumented

`foundRegion`
    On return, a 16-bit signed integer containing an international region code; see `Localization Codes`.

`theString`
    The found string.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

ITextRemoveString

Undocumented

```
OSErr ITextRemoveString (
   QTAtomContainer container,
   QTAtom parentAtom,
   RegionCode theRegionCode,
   long flags
);
```

Parameters

container

Undocumented

parentAtom

Undocumented

theRegionCode

A 16-bit signed integer containing an international region code; see `Localization Codes`.

flags

Flags (see below) that modify the process. See these constants:

`kITextRemoveEverythingBut`

`kITextRemoveLeaveSuggestedAlternate`

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

LoadMediaIntoRam

Loads a media's data into memory.

```
OSErr LoadMediaIntoRam (
   Media theMedia,
   TimeValue time,
   TimeValue duration,
   long flags
);
```

Parameters

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (page 1630) and `GetTrackMedia` (page 1612). See `Media Identifiers`.

time

The starting time of the media segment to load. This time value must be expressed in the media's time coordinate system.

duration
: The length of the segment to load. Use `GetMediaDuration` (page 1576) to determine the length of the entire media. Note that the media handler may load more data than you specify if the media data was added in larger pieces.

flags
: Flags that give you explicit control over what is loaded into memory and how long to keep it around. See `RAM Loading Flags`. You can set these flags in any combination.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
The exact behavior of `LoadMediaIntoRam` is dependent on the media handler.

Special Considerations

If `LoadMediaIntoRam` fails because it is out of memory, no data is purged.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

LoadMovieIntoRam

Loads a movie's data into memory.

```
OSErr LoadMovieIntoRam (
   Movie theMovie,
   TimeValue time,
   TimeValue duration,
   long flags
);
```

Parameters

theMovie
: The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

time
: The starting time of the movie segment to load.

duration
: The length of the segment to load. Use `GetMovieDuration` (page 211) to determine the length of the entire movie. Note that the Movie Toolbox may load more data than you specify due to the way the data is loaded.

flags
: Flags that give you explicit control over what is loaded into memory and how long to keep it around. See `RAM Loading Flags`. You can set these flags in any combination.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CompressMovies

DigitizerShell

DragAndDrop Shell

MovieGWorlds

QT Internals

Declared In
`Movies.h`

LoadTrackIntoRam

Loads a track's data into memory.

```
OSErr LoadTrackIntoRam (
   Track theTrack,
   TimeValue time,
   TimeValue duration,
   long flags
);
```

Parameters
`theTrack`
: The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

`time`
: The starting time of the track segment to load. You must specify this time value in the movie's time coordinate system.

`duration`
: The length of the segment to load. Use `GetTrackDuration` (page 1607) to determine the length of the entire movie. Note that the media handler may load more data than you specify.

`flags`
: Flags that give you explicit control over what is loaded into memory and how long to keep it around. See `RAM Loading Flags`. You can set these flags in any combination.

Return Value
If the track does not fit, the function returns an error. See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code

qttext

qttext.win

Declared In

`Movies.h`

Media3DGetCameraAngleAspect

Deprecated.

```
ComponentResult Media3DGetCameraAngleAspect (
   MediaHandler mh,
   QTFloatSingle *fov,
   QTFloatSingle *aspectRatioXToY
);
```

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

Media3DGetCameraData

Deprecated.

```
ComponentResult Media3DGetCameraData (
   MediaHandler mh,
   void *cameraData
);
```

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

Media3DGetCameraRange

Deprecated.

```
ComponentResult Media3DGetCameraRange (
   MediaHandler mh,
   void *tQ3CameraRange
);
```

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

Media3DGetCurrentGroup

Deprecated.

```
ComponentResult Media3DGetCurrentGroup (
   MediaHandler mh,
   void *group
);
```

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

Media3DGetNamedObjectList

Deprecated.

```
ComponentResult Media3DGetNamedObjectList (
   MediaHandler mh,
   QTAtomContainer *objectList
);
```

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

Media3DGetRendererList

Deprecated.

```
ComponentResult Media3DGetRendererList (
   MediaHandler mh,
   QTAtomContainer *rendererList
);
```

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

Media3DGetViewObject

Deprecated.

```
ComponentResult Media3DGetViewObject (
   MediaHandler mh,
   void *tq3viewObject
);
```

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

Media3DRotateNamedObjectTo

Deprecated.

```
ComponentResult Media3DRotateNamedObjectTo (
   MediaHandler mh,
   char *objectName,
   Fixed xDegrees,
   Fixed yDegrees,
   Fixed zDegrees
);
```

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

Media3DScaleNamedObjectTo

Deprecated.

```
ComponentResult Media3DScaleNamedObjectTo (
   MediaHandler mh,
   char *objectName,
   Fixed xScale,
   Fixed yScale,
   Fixed zScale
);
```

Return Value

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

Media3DSetCameraAngleAspect

Deprecated.

```
ComponentResult Media3DSetCameraAngleAspect (
   MediaHandler mh,
   QTFloatSingle fov,
   QTFloatSingle aspectRatioXToY
);
```

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

Media3DSetCameraData

Deprecated.

```
ComponentResult Media3DSetCameraData (
   MediaHandler mh,
   void *cameraData
);
```

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

Media3DSetCameraRange

Deprecated.

```
ComponentResult Media3DSetCameraRange (
   MediaHandler mh,
   void *tQ3CameraRange
);
```

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

Media3DTranslateNamedObjectTo

Deprecated.

```
ComponentResult Media3DTranslateNamedObjectTo (
   MediaHandler mh,
   char *objectName,
   Fixed x,
   Fixed y,
   Fixed z
);
```

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

MovieMediaGetChildDoMCActionCallback

Undocumented

```
ComponentResult MovieMediaGetChildDoMCActionCallback (
   MediaHandler mh,
   DoMCActionUPP *doMCActionCallbackProc,
   long *refcon
);
```

Parameters

mh

A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

doMCActionCallbackProc

A pointer to a Universal Procedure Pointer that accesses a `DoMCActionProc` callback.

refcon

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

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 4.1.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

MovieMediaGetChildMovieDataReference

Undocumented

```
ComponentResult MovieMediaGetChildMovieDataReference (
   MediaHandler mh,
   QTAtomID dataRefID,
   short dataRefIndex,
   OSType *dataRefType,
   Handle *dataRef,
   QTAtomID *dataRefIDOut,
   short *dataRefIndexOut
);
```

Parameters

mh

A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

dataRefID

Undocumented

dataRefIndex

Undocumented

dataRefType

Undocumented

dataRef

Undocumented

dataRefIDOut

Undocumented

dataRefIndexOut

Undocumented

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 4.1.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

MovieMediaGetCurrentMovieProperty

Retrieves current properties from a media handler's movie.

```
ComponentResult MovieMediaGetCurrentMovieProperty (
   MediaHandler mh,
   OSType whichProperty,
   void *value
);
```

Parameters

mh

A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

whichProperty

A constant (see below) that designates the property to be retrieved. See these constants:

`kMoviePropertyDuration`

`kMoviePropertyTimeScale`

`kMoviePropertyTime`

`kMoviePropertyNaturalBounds`

`kMoviePropertyMatrix`

`kMoviePropertyTrackList`

value

A pointer to the returned property value.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

MovieMediaGetCurrentTrackProperty

Retrieves the media type property from a media handler's track.

```
ComponentResult MovieMediaGetCurrentTrackProperty (
   MediaHandler mh,
   long trackID,
   OSType whichProperty,
   void *value
);
```

Parameters

mh

A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

trackID

The ID value of the track for this operation.

whichProperty

A constant (see below) that designates the property to be retrieved. Only the track's media type property constant is currently defined. See these constants:

`kTrackPropertyMediaType`

value

A pointer to the returned property value.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

MovieMediaGetDoMCActionCallback

Gets a DoMCActionProc callback for a media.

```
ComponentResult MovieMediaGetDoMCActionCallback (
   MediaHandler mh,
   DoMCActionUPP *doMCActionCallbackProc,
   long *refcon
);
```

Parameters

mh

A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

doMCActionCallbackProc

A pointer to a Universal Procedure Pointer that accesses a `DoMCActionProc` callback.

refcon

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

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 4.1.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

MovieMediaLoadChildMovieFromDataReference

Undocumented

```
ComponentResult MovieMediaLoadChildMovieFromDataReference (
   MediaHandler mh,
   QTAtomID dataRefID
);
```

Parameters

mh

A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

dataRefID

Undocumented

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 4.1.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

MovieMediaSetChildMovieDataReference

Undocumented

```
ComponentResult MovieMediaSetChildMovieDataReference (
   MediaHandler mh,
   QTAtomID dataRefID,
   OSType dataRefType,
   Handle dataRef
);
```

Parameters

`mh`
A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

`dataRefID`
Undocumented

`dataRefType`
Undocumented

`dataRef`
Undocumented

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

MoviesTask

Services active movies.

```
void MoviesTask (
   Movie theMovie,
   long maxMilliSecToUse
);
```

Parameters

`theMovie`
The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400). If you set this parameter to `NIL`, the Movie Toolbox services all of your active movies.

maxMilliSecToUse

Determines the maximum number of milliseconds that `MoviesTask` can work before returning. If this parameter is 0, `MoviesTask` services every active movie exactly once and then returns. If the parameter is nonzero, `MoviesTask` services as many movies as it can in the allotted time before returning. Once the `MoviesTask` function starts servicing a movie, it cannot stop until it has completely met the requirements of the movie. Consequently, the `MoviesTask` function may execute for a longer time than that specified in `maxMilliSecToUse`. However, the function does not start servicing a new movie if the time specified by `maxMilliSecToUse` has elapsed. The preferred way to use `MoviesTask` is to set the `maxMilliSecToUse` parameter to 0; however, if you just want to play one movie, you can call `MoviesTask` on that one. If your rate is 0, `MoviesTask` draws that frame and no other.

Return Value

You can access error returns from this function through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). See `Error Codes`.

Discussion

You should call `MoviesTask` as often as possible from your application's main event loop. Note that you should call this function after you have performed your own event processing. `MoviesTask` services only active movies, and only enabled tracks within those active movies.

Special Considerations

Note that the `MoviesTask` function services only your movies. Your application must give other applications the opportunity to call `MoviesTask` for their movies.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CompressMovies

Graphic Import-Export

MovieGWorlds

vrscript

vrscript.win

Declared In

`Movies.h`

NewCallBack

Creates a new callback event.

```
QTCallBack NewCallBack (
   TimeBase tb,
   short cbType
);
```

Parameters

tb

The callback event's time base. You obtain this identifier from `NewTimeBase` (page 261).

cbType

Constants (see below) that specify when the callback event is to be invoked. The value of this field governs how the Movie Toolbox interprets the data supplied in the param1, param2, and param3 parameters to the `CallMeWhen` (page 179) function. In addition, if the high-order bit of the `cbType` parameter is set to 1 (this bit is defined by the `callBackAtInterrupt` flag), the event can be invoked at interrupt time. See these constants:

`callBackAtTime`

`callBackAtRate`

`callBackAtTimeJump`

`callBackAtExtremes`

`callBackAtInterrupt`

Return Value

A pointer to a `CallBackRecord` structure containing the new callback event.

Special Considerations

The callback event created is not active until you schedule it by calling the `CallMeWhen` (page 179) function. You must not call this function at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qtbigscreen

qtbigscreen.win

Show Movie

SimpleCocoaMovie

SimpleCocoaMovieQT

Declared In

`Movies.h`

NewMovie

Creates a new movie in memory.

```
Movie NewMovie (
   long flags
);
```

Parameters

flags

Flags (see below) that specify control information for the new movie. Be sure to set unused flags to 0.

Return Value

The identifier for the new movie. If `NewMovie` fails, the returned identifier is set to `NIL`. You can use `GetMoviesError` (page 221) to obtain the error result, or `noErr` if there was no error. See `Error Codes`.

Discussion

You can use `NewMovie` to create a new empty movie, which contains no tracks. The Movie Toolbox initializes the data structures for the new movie. Your application assigns the data to the movie by calling the functions that are described in `NewMovieTrack` (page 1628).

The Movie Toolbox sets many movie characteristics to default values. If you want to change these defaults, your application must call other Movie Toolbox functions. For example, the Movie Toolbox sets the movie's graphics world to the one that is active when you call `NewMovie`. To change the graphics world for the new movie, your application should use `SetMovieGWorld` (page 290). The default QuickTime movie time scale is 600 units per second; however, this number may change in the future. The default time scale was chosen because it is convenient for working with common video frame rates of 30, 25, 24, 15, 12, 10, and 8.

Special Considerations

The Movie Toolbox automatically sets the movie's graphics world based on the current graphics port. Be sure that your application's graphics port is valid before you call this function, even if the movie is sound-only; you can use `GetGWorld` to check for a valid port, or you can use `NewGWorld` to create a port. The graphics port must remain valid for the life of the movie or until you set another valid graphics port for the movie using `SetMovieGWorld` (page 290).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Graphic Import-Export

mfc.win

qtdataref

qtdataref.win

SoundPlayer.win

Declared In

`Movies.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`.

outputPropertyCount
: The number of properties in the array passed in `outputProperties`.

outputProperties
: A pointer to a property array to receive output parameters. See `QTNewMoviePropertyElement`. 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`.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
LiveVideoMixer2
LiveVideoMixer3
QTPixelBufferVCToCGImage
SimpleAudioExtraction
SimpleHIMovieViewPlayer

Declared In
`Movies.h`

NewTimeBase

Obtains a new time base.

```
TimeBase NewTimeBase (
   void
);
```

Return Value
The ID of the new time base.

Discussion
This function sets the rate of the time base to 0, the start time to its minimum value, the time value to 0, and the stop time to its maximum value. The function assigns the default clock component to the new time base.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qteffects.win

QTMusicToo

qtshoweffect

VideoProcessing

vrscript.win

Declared In
`Movies.h`

PlayMoviePreview

Plays a movie's preview.

```
void PlayMoviePreview (
   Movie theMovie,
   MoviePreviewCallOutUPP callOutProc,
   long refcon
);
```

Parameters

theMovie
: The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

callOutProc
: A pointer to a `MoviePreviewCallOutProc` callback in your application. The Movie Toolbox calls this function repeatedly while the movie preview is playing. You can use this function to stop the preview. If you don't want to assign a function, set this parameter to `NIL`.

refcon
: A reference constant that the Movie Toolbox passes to your callback. Use this parameter to point to a data structure containing any information your callback needs.

Return Value
You can access error returns from this function through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). See `Error Codes`.

Discussion
This function sets the movie into preview mode, plays the movie preview, sets the movie back to normal playback mode, and returns to your application. The Movie Toolbox plays the preview in the movie's graphics world. Note that if you call the `GetMovieActiveSegment` (page 205) function from within your movie callout function, the Movie Toolbox will have changed the active movie segment to be the preview segment of the movie. The Movie Toolbox restores the active segment when the preview is done playing.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qtinfo
qtinfo.win

Declared In
`Movies.h`

PrePrerollMovie

Sets up any necessary network connections to receive streaming content.

```
OSErr PrePrerollMovie (
   Movie m,
   TimeValue time,
   Fixed rate,
   MoviePrePrerollCompleteUPP proc,
   void *refcon
);
```

Parameters

m
: The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

time
: The starting time of the movie segment to play.

rate
: The rate at which you anticipate playing the movie. You specify the movie rate as a 32-bit, fixed-point number. Positive integers indicate forward rates and negative integers indicate reverse rates.

proc
: The `MoviePrePrerollCompleteProc` callback you want called when pre-prerolling is complete. If a completion `proc` is specified, `PrePrerollMovie` operates asynchronously. You must call `MoviesTask` periodically during asynchronous operation. If no completion `proc` is specified, `PrePrerollMovie` operates synchronously.

refcon
: A reference constant that is passed to your callback. Use this parameter to point to a data structure containing any information your callback needs.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
Before a movie is played, it is normally prerolled. During preroll, the Movie Toolbox tells the appropriate media handlers to load the movie data, allocate sound channels, start up image-decompression sequences, and so on. Before a movie that contains streaming content is prerolled, it must be pre-prerolled. This sets up any necessary network connections between the client and the server. If a movie contains streaming content (one or more `'strm'` tracks), you must call this function before calling `PrerollMovie` (page 264). If the movie does not contain streaming content, calling this function has no effect. If your application calls `PrerollMovie`, it should always call this function first. If you play movies using a movie controller, you don't need to preroll or pre-preroll the movie explicitly; it is done for you automatically. If a completion `proc` is specified in the `proc` parameter, this function operates asynchronously; it returns almost immediately and calls the completion `proc` when pre-prerolling is complete.

Special Considerations

You must call `MoviesTask` (page 257) periodically to grant time for pre-prerolling during asynchronous operation. If no completion `proc` is specified, this function operates synchronously; the function will not return until pre-prerolling is complete. This can take a long time, particularly if a dial-up network connection must be established.

Version Notes

Introduced in QuickTime 4. Beginning with QuickTime 4, your application should call this function any time it calls `PrerollMovie` (page 264).

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qtfullscreen

qtfullscreen.win

vrscript

vrscript.win

Declared In

`Movies.h`

PrerollMovie

Prepares a portion of a movie for playback.

```
OSErr PrerollMovie (
   Movie theMovie,
   TimeValue time,
   Fixed Rate
);
```

Parameters

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

time

The starting time of the movie segment to play.

Rate

The rate at which you anticipate playing the movie. You specify the movie rate as a 32-bit, fixed-point number. Positive integers indicate forward rates and negative integers indicate reverse rates.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

When your application calls `PrerollMovie`, the Movie Toolbox tells the appropriate media handlers to prepare to play the movie. The media handlers may then load the movie data and perform any other necessary preparations to play the movie, such as allocating sound channels and starting up image-decompression sequences. In this manner, you can eliminate playback stutter when the movie starts playing.

If your application uses QuickTime's Movie Toolbox to play back movies, there are two choices for how to preroll the movie. Like the movie controller, the Movie Toolbox provides a single function call, `StartMovie` (page 332), which will both preroll the movie and start it playing. Unlike the movie controller, the Movie Toolbox function doesn't allow you to specific the rate to play the movie at, but instead assumes the movie's preferred rate.

Calling `StartMovie`, just like the movie controller's preroll and play action, first prerolls the movie and then sets it playing. If your application requires more control, the Movie Toolbox provides lower level functions that give you more control:

```
// PrerollMovie coding example
StartMovie(theMovie);

TimeValue timeNow;
Fixed playRate;
timeNow =GetMovieTime(theMovie, NIL);
playRate =GetMoviePreferredRate(theMovie);
PrePrerollMovie(theMovie, timeNow, playRate, NIL, NIL);
PrerollMovie(theMovie, timeNow, playRate);
SetMovieRate(theMovie, playRate);
```

Special Considerations

You should always call `PrePrerollMovie` (page 263) before calling this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

DigitizerShell

LiveVideoMixer2

LiveVideoMixer3

MovieBrowser

MovieGWorlds

Declared In

`Movies.h`

PutMovieForDataRefIntoHandle

Puts a self-contained movie into a handle.

```
OSErr PutMovieForDataRefIntoHandle (
   Movie theMovie,
   Handle dataRef,
   OSType dataRefType,
   Handle publicMovie
);
```

Parameters

theMovie
: The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

dataRef
: A handle to the storage in which the movie will be written.

dataRefType
: The data reference type. See `Data References`.

publicMovie
: The handle that is to receive the new movie resource. The function resizes the handle if necessary.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
If the data reference and data reference type is passed, all media references to the same storage are converted to self-references in the resulting public movie handle. This `'moov'` atom can be then written to the storage.

Version Notes
Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`Movies.h`

PutMovieIntoDataFork

Stores a movie in the data fork of a given file.

```
OSErr PutMovieIntoDataFork (
   Movie theMovie,
   short fRefNum,
   long offset,
   long maxSize
);
```

Parameters

theMovie
: The movie to be stored in the data fork of an atom. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

fRefNum

A file reference number for the data fork of the given file. You pass in an open write path in the `fRefNum` parameter.

offset

Indicates where the movie should be written.

maxSize

The largest number of bytes that may be written.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

PutMovieIntoDataFork64

Provides a 64-bit version of PutMovieIntoDataFork.

```
OSErr PutMovieIntoDataFork64 (
   Movie theMovie,
   long fRefNum,
   const wide *offset,
   unsigned long maxSize
);
```

Parameters

theMovie

A movie identifier. Your application obtains this identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

fRefNum

A file reference number for the data fork of the given file. You pass in an open write path in the `fRefNum` parameter.

offset

Pointer to a 64-bit value that indicates where the movie should be written.

maxSize

The largest number of bytes that may be written.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 4. Superseded in QuickTime 6 by `PutMovieIntoStorage` (page 268).

Availability

Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

PutMovieIntoHandle

Creates a new movie resource.

```
OSErr PutMovieIntoHandle (
   Movie theMovie,
   Handle publicMovie
);
```

Parameters

theMovie
: The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

publicMovie
: The handle that is to receive the new movie resource. The function resizes the handle if necessary.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
Use this handle to store a QuickTime movie in a specialized storage format.

Special Considerations
Note that you cannot use this new movie with other Movie Toolbox functions, except for `NewMovieFromHandle` (page 1400).

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CompressMovies

DragAndDrop Shell

QTAudioExtractionPanel

QTExtractAndConvertToAIFF

ThreadsExportMovie

Declared In
`Movies.h`

PutMovieIntoStorage

Writes a movie to a storage location managed by a data handler.

```
OSErr PutMovieIntoStorage (
   Movie theMovie,
   DataHandler dh,
   const wide *offset,
   unsigned long maxSize
);
```

Parameters

theMovie
: The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

dh
: A data handler for the data fork of the storage container. You pass an open write path in this parameter.

offset
: A pointer to a value that indicates where the movie should be written in the container.

maxSize
: The largest number of bytes that may be written.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

If you are writing a custom data handler, make sure it supports `DataHGetDataRef` (page 780). It must also support `DataHWrite64` (page 819), or `DataHWrite` (page 817) if 64-bit offsets are not supported.

Version Notes

Introduced in QuickTime 6. This function supersedes `PutMovieIntoDataFork64` (page 267).

Availability

Available in Mac OS X v10.2 and later.

Declared In

`Movies.h`

QTAudioContextCreateForAudioDevice

Creates a QTAudioContext object that encapsulates a connection to a CoreAudio output device.

```
OSStatus QTAudioContextCreateForAudioDevice (
   CFAllocatorRef allocator,
   CFStringRef audioDeviceUID,
   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` (page 260), 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` (page 260), you can release the object since these APIs will retain it for their own use.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

QTSetMovieAudioDevice

Declared In

`Movies.h`

QTGetTimeUntilNextTask

Reports the duration until the next time QuickTime needs to run a task.

```
OSErr QTGetTimeUntilNextTask (
   long *duration,
   long scale
);
```

Parameters

duration

A pointer to the duration until the next time QuickTime needs access to the processor. If the returned duration is 0, QuickTime needs to run a task immediately.

scale

The time scale in which to express the returned duration. For example, pass 60 if you want the duration value expressed in ticks (60ths of a second).

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

Periodically, applications have to give processing time to QuickTime by calling a function such as `MCIsPlayerEvent` (page 1204). Instead of routinely calling `MCIsPlayerEvent` 10 to 20 times per second, you can call `QTGetTimeUntilNextTask` to determine when QuickTime next needs access to the processor. The result is a more efficent use of processor resources. To handle cases when QuickTime may need to run a task earlier than projected by this function, you can install a `QTNextTaskNeededSoonerCallbackProc` callback.

Version Notes
Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
QTCarbonCoreImage101

qtshellCEvents

qtshellCEvents.win

VideoProcessing

Declared In
`Movies.h`

QTGetWallClockTimeBase

Returns the system's real-time time base.

```
OSErr QTGetWallClockTimeBase (
   TimeBase *wallClockTimeBase
);
```

Parameters

wallClockTimeBase
A pointer to the wall clock's time base.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`Movies.h`

QTIdleManagerClose

Closes the Mac OS Idle Manager.

```
OSErr QTIdleManagerClose (
   IdleManager im
);
```

Parameters

im
A pointer to the opaque data structure that was returned by `QTIdleManagerOpen` (page 273).

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
Your application should call this function after it no longer needs access to the Idle Manager.

Version Notes
Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`Movies.h`

QTIdleManagerGetNextIdleTime

Retrieves the next idle time known to the Idle Manager.

```
OSErr QTIdleManagerGetNextIdleTime (
   IdleManager im,
   TimeRecord *nextIdle
);
```

Parameters

`im`
A pointer to an opaque data structure that belongs to the Mac OS Idle Manager. You get this pointer by calling `QTIdleManagerOpen` (page 273).

`nextIdle`
A pointer to the next idle time.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`Movies.h`

QTIdleManagerNeedsAnIdle

Tells the Idle Manager whether an idle will be required.

```
OSErr QTIdleManagerNeedsAnIdle (
   IdleManager im,
   Boolean *needsOne
);
```

Parameters

im

A pointer to an opaque data structure that belongs to the Mac OS Idle Manager. You get this pointer by calling `QTIdleManagerOpen` (page 273).

needsOne

Pass a pointer to a variable; on return, TRUE means that an idle will be required, FALSE means no idle will be required.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 6.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`Movies.h`

QTIdleManagerOpen

Opens the Mac OS Idle Manager.

```
IdleManager QTIdleManagerOpen (
   void
);
```

Return Value

A pointer to an opaque data structure that belongs to the Mac OS Idle Manager.

Discussion

You must call this function before using the Mac OS Idle Manager.

Version Notes

Introduced in QuickTime 6.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`Movies.h`

QTIdleManagerSetNextIdleTime

Informs the idle manager of the next required idle time.

```
OSErr QTIdleManagerSetNextIdleTime (
   IdleManager im,
   TimeRecord *nextIdle
);
```

Parameters

im

A pointer to an opaque data structure that belongs to the Mac OS Idle Manager. You get this pointer by calling `QTIdleManagerOpen` (page 273).

nextIdle

A pointer to the time of the next required idle.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

If a media handler needs to call this function, you must do wallclock time calculations. That means you may need to call `QTGetWallClockTimeBase` (page 271) and `ConvertTime` (page 183) to convert from track time or media time to wallclock time, plus `ConvertTimeScale` (page 183) to convert to the timescale you like to work in.

Version Notes

Introduced in QuickTime 6.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`Movies.h`

QTIdleManagerSetNextIdleTimeDelta

Informs the idle manager of the time from the currently set idle time to the next idle time required after it.

```
OSErr QTIdleManagerSetNextIdleTimeDelta (
   IdleManager im,
   TimeValue duration,
   TimeScale scale
);
```

Parameters

im

A pointer to an opaque data structure that belongs to the Mac OS Idle Manager. You get this pointer by calling `QTIdleManagerOpen` (page 273).

duration

The time from the current idle time to the next one.

scale

The time scale in which the duration is expressed.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

This routine lets you pass in a duration and a scale and it gives you a single idle. For example, if you need an idle a half second from now, you can pass in a duration of 500 and a scale of 1000, or a duration of 1 and scale of 2. This will get you one idle 0.5 seconds from now.

Special Considerations

Every time you get idled, you need to call this function again to set your next idle. If you don't, QuickTime will assume a default duration to the next idle of 0 and you'll be idled all the time.

Version Notes

Introduced in QuickTime 6.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`Movies.h`

QTIdleManagerSetNextIdleTimeNever

Sets the next idle time indefinitely in the future.

```
OSErr QTIdleManagerSetNextIdleTimeNever (
   IdleManager im
);
```

Parameters

`im`

A pointer to an opaque data structure that belongs to the Mac OS Idle Manager. You get this pointer by calling `QTIdleManagerOpen` (page 273).

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 6.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`Movies.h`

QTIdleManagerSetNextIdleTimeNow

Requests an idle as soon as possible.

```
OSErr QTIdleManagerSetNextIdleTimeNow (
   IdleManager im
);
```

Parameters

im

A pointer to an opaque data structure that belongs to the Mac OS Idle Manager. You get this pointer by calling `QTIdleManagerOpen` (page 273).

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 6.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`Movies.h`

QTIdleManagerSetParent

Sets the parent of an Idle Manager instance.

```
OSErr QTIdleManagerSetParent (
   IdleManager im,
   IdleManager parent
);
```

Parameters

im

A pointer to an opaque data structure that belongs to the Mac OS Idle Manager. You get this pointer by calling `QTIdleManagerOpen` (page 273).

parent

A pointer to a different Idle Manager data structure. You get this pointer also by calling `QTIdleManagerOpen` (page 273).

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 6.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`Movies.h`

QTInstallNextTaskNeededSoonerCallback

Installs a QTNextTaskNeededSoonerCallbackProc callback.

```
OSErr QTInstallNextTaskNeededSoonerCallback (
   QTNextTaskNeededSoonerCallbackUPP callbackProc,
   TimeScale scale,
   unsigned long flags,
   void *refcon
);
```

Parameters

callbackProc

A Universal Procedure Pointer to a `QTNextTaskNeededSoonerCallbackProc` callback.

scale

The time scale that QuickTime will use when reporting the duration until the next time QuickTime needs to be called, via `QTGetTimeUntilNextTask` (page 270).

flags

Unused; set to 0.

refcon

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

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

This routine installs a callback procedure that specifies when QuickTime next needs to be tasked. The callback procedure may be called at interrupt time or from another Mac OS X thread, so you must be careful not to cause race conditions. You can install or uninstall multiple callback procedures if necessary; they will be called in sequence. You can also install the same callback multiple times with different `refcon` values, in which case it will be called once with each `refcon` value.

Version Notes

Introduced in QuickTime 6.

Availability

Available in Mac OS X v10.2 and later.

Related Sample Code

QTCarbonCoreImage101

qtshellCEvents

qtshellCEvents.win

VideoProcessing

Declared In

`Movies.h`

QTParseTextHREF

Undocumented

```
OSErr QTParseTextHREF (
   char *href,
   SInt32 hrefLen,
   QTAtomContainer inContainer,
   QTAtomContainer *outContainer
);
```

Parameters

href
    A pointer to an HREF string.

hrefLen
    The length of the HREF string.

inContainer
    Undocumented

outContainer
    Undocumented

Return Value
You can access Movie Toolbox error returns through GetMoviesError (page 221) and GetMoviesStickyError (page 222), as well as in the function result. See Error Codes.

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

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 `kQTSoundDescriptionKind_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. Use these constants:

```
enum {
  kQTSoundDescriptionKind_Movie_Version1 = 'mvv1',
  kQTSoundDescriptionKind_Movie_Version2 = 'mvv2',
  kQTSoundDescriptionKind_Movie_LowestPossibleVersion = 'mvlo',
  kQTSoundDescriptionKind_Movie_AnyVersion = 'mvny'
};
```

Availability

Available in Mac OS X v10.3 and later.

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 (see Discussion, below).

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.

Discussion

The value of `inRequestedKind` can be taken from these values:

```
enum {
  kQTSoundDescriptionKind_Movie_Version1 = 'mvv1',
  kQTSoundDescriptionKind_Movie_Version2 = 'mvv2',
  kQTSoundDescriptionKind_Movie_LowestPossibleVersion = 'mvlo',
  kQTSoundDescriptionKind_Movie_AnyVersion = 'mvny'
};
```

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

QTExtractAndConvertToMovieFile

SCAudioCompress

WhackedTV

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.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
CaptureAndCompressIPBMovie

WhackedTV

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.

Availability
Available in Mac OS X v10.3 and later.

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.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
SCAudioCompress

Declared In
`Movies.h`

QTTextToNativeText

Undocumented

```
OSErr QTTextToNativeText (
   Handle theText,
   long encoding,
   long flags
);
```

Parameters

theText
    Undocumented

encoding

Undocumented

flags

Flags (see below) that define the `text` atom type. See these constants:

`kITextAtomType`

`kITextStringAtomType`

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

QTUninstallNextTaskNeededSoonerCallback

Removes a QTNextTaskNeededSoonerCallbackProc callback.

```
OSErr QTUninstallNextTaskNeededSoonerCallback (
   QTNextTaskNeededSoonerCallbackUPP callbackProc,
   void *refcon
);
```

Parameters

callbackProc

A Universal Procedure Pointer to a `QTNextTaskNeededSoonerCallbackProc` callback that you installed by a previous call to `QTInstallNextTaskNeededSoonerCallback` (page 277).

refcon

A pointer to the reference constant that you passed when the callback was installed.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

You pass this routine both a pointer to a callback procedure and a pointer to its reference constant, so you can uninstall one instance of a callback that you installed more than once with different `refcon` values.

Version Notes

Introduced in QuickTime 6.

Availability

Available in Mac OS X v10.2 and later.

Related Sample Code

QTCarbonCoreImage101

VideoProcessing

Declared In
`Movies.h`

RemoveCallBackFromTimeBase

Removes a callback event from the list of scheduled callback events.

```
OSErr RemoveCallBackFromTimeBase (
   QTCallBack cb
);
```

Parameters

cb

The callback event for the operation. Your clock component obtains this value from the parameters passed to your `ClockCallMeWhen` (page 453) function.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
Your clock component should call this function when your `ClockCancelCallBack` (page 454) function determines that your component can cancel the callback event.

Special Considerations

Your component should call this function only for callback events that were successfully added to the schedule with `AddCallBackToTimeBase` (page 178).

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

SetMovieActive

Activates or deactivates a movie.

```
void SetMovieActive (
   Movie theMovie,
   Boolean active
);
```

Parameters

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

active

Activates or deactivates the `movie`. Set this parameter to TRUE to activate the `movie`; set this parameter to FALSE to deactivate the movie.

Return Value

You can access error returns from this function through GetMoviesError (page 221) and GetMoviesStickyError (page 222). See Error Codes.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

BurntTextSampleCode

QTCarbonShell

qtcontroller

qtshellCEvents.win

vrcursors

Declared In

Movies.h

SetMovieActiveSegment

Defines a movie's active segment.

```
void SetMovieActiveSegment (
   Movie theMovie,
   TimeValue startTime,
   TimeValue duration
);
```

Parameters

theMovie
: The movie for this operation. Your application obtains this movie identifier from such functions as NewMovie (page 259), NewMovieFromFile (page 1398), and NewMovieFromHandle (page 1400).

startTime
: A time value specifying the starting point of the active segment. Set this parameter to -1 to make the entire movie active. In this case, the SetMovieActiveSegment function ignores the duration parameter.

duration
: A time value that specifies the duration of the active segment. If you are making the entire movie active (by setting the startTime parameter to -1), the Movie Toolbox ignores this parameter.

Return Value

You can access error returns from this function through GetMoviesError (page 221) and GetMoviesStickyError (page 222). See Error Codes.

Discussion

Your application defines the active segment by specifying the starting time and duration of the segment. These values must be expressed in the movie's time coordinate system. By default, the entire movie is active.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

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. .

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
QTSetMovieAudioDevice

Declared In
`Movies.h`

SetMovieBox

Sets a movie's boundary rectangle.

```
void SetMovieBox (
   Movie theMovie,
   const Rect *boxRect
);
```

Parameters

theMovie
The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

boxRect
A pointer to a rectangle that contains the coordinates of the new boundary rectangle.

Return Value
You can access error returns from this function through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). See `Error Codes`.

Discussion

The Movie Toolbox changes the rectangle by modifying the translation and scale values of the movie's matrix to accommodate the new boundary rectangle.

The movie box might not have its upper-left corner set at (0,0) in its display window when the movie is first loaded. Consequently, your application may need to adjust the position of the movie box so that it appears in the appropriate location within your application's document window. If you don't reset the movie position, the movie might not be visible when it starts playing. The following sample code demonstrates how to do this:

```
//Zeroing the boundary rectangle with SetMovieBox
GetMovieBox (movie, &movieBox);
OffsetRect (&movieBox, -movieBox.left, -movieBox.top);
SetMovieBox (movie, &movieBox);
```

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

MovieGWorlds

vrmovies

vrmovies.win

vrscript

vrscript.win

Declared In

`Movies.h`

SetMovieClipRgn

Establishes a movie's clipping region.

```
void SetMovieClipRgn (
   Movie theMovie,
   RgnHandle theClip
);
```

Parameters

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

theClip

A handle to the movie's clipping region. The Movie Toolbox makes a copy of this region. Your application must dispose of the region referred to by this parameter when you are done with it. Set this parameter to `NIL` to disable clipping for the movie.

Return Value

You can access error returns from this function through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). See `Error Codes`.

Discussion
The clipping region is saved with the movie.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CompressMovies
ConvertToMovieJr
qtcompress
qtcompress.win
VideoProcessing

Declared In
`Movies.h`

SetMovieDisplayClipRgn

Establishes a movie's current display clipping region.

```
void SetMovieDisplayClipRgn (
   Movie theMovie,
   RgnHandle theClip
);
```

Parameters

theMovie
: The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

theClip
: A handle to the movie's display clipping region as a `MacRegion` structure. The Movie Toolbox makes a copy of this region. Your application must dispose of the region referred to by this parameter when you are done with it. Set this parameter to `NIL` to disable a movie's clipping region.

Return Value
You can access error returns from this function through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). See `Error Codes`.

Discussion
The display clipping region is not saved with the movie. You can access error returns from this function through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

SetMovieDrawingCompleteProc

Assigns a drawing-complete function to a movie.

```
void SetMovieDrawingCompleteProc (
   Movie theMovie,
   long flags,
   MovieDrawingCompleteUPP proc,
   long refCon
);
```

Parameters

theMovie

The movie for this operation.

flags

Contains flags (see below) that control when your drawing complete function is called. See these constants:

`movieDrawingCallWhenChanged`

`movieDrawingCallAlways`

proc

A pointer to your `MovieDrawingCompleteProc` callback. Set this parameter to `NIL` if you want to remove your callback.

refCon

The reference constant you supplied when your application called your callback. Use this parameter to point to a data structure containing any information your callback needs.

Return Value

You can access error returns from this function through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). See `Error Codes`.

Discussion

The Movie Toolbox calls this function based upon guidelines you establish when you assign the function to the movie.

Special Considerations

Some media handlers may take less efficient playback paths when a drawing-complete function is used, so it should be used only when absolutely necessary.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

ASCIIMoviePlayerSample

bMoviePaletteCocoa

MovieGWorlds

OpenGLMovieQT

VideoProcessing

Declared In

`Movies.h`

SetMovieGWorld

Establishes a movie's display coordinate system by setting the graphics world for displaying the movie.

```
void SetMovieGWorld (
   Movie theMovie,
   CGrafPtr port,
   GDHandle gdh
);
```

Parameters

theMovie
: The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

port
: Points to the movie's `CGrafPort` structure or graphics world. Set this parameter to `NIL` to use the current graphics port.

gdh
: A handle to the movie's `GDevice` structure. Set this parameter to `NIL` to use the current device. If the `port` parameter specifies a graphics world, set this parameter to `NIL` to use that graphics world's graphics device.

Return Value

You can access error returns from this function through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). See `Error Codes`.

Special Considerations

When you use this function, the Movie Toolbox remembers the current background color and background pattern. These are used for erasing in the default movie uncover function; see `SetMovieCoverProcs` (page 1482).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

MovieGWorlds

QTCarbonShell

vrmovies

vrscript

vrscript.win

Declared In

`Movies.h`

SetMovieMasterClock

Assigns a clock component to a movie.

```
void SetMovieMasterClock (
   Movie theMovie,
   Component clockMeister,
   const TimeRecord *slaveZero
);
```

Parameters

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as NewMovie (page 259), NewMovieFromFile (page 1398), and NewMovieFromHandle (page 1400).

clockMeister

The clock component to be assigned to this movie. Your application can obtain this component identifier from FindNextComponent.

slaveZero

A pointer to the time, in the clock's time scale, that corresponds to a 0 time value for the movie. This parameter allows you to set an offset between the clock component and the time base of the movie. Set this parameter to NIL if there is no offset.

Return Value

You can access error returns from this function through GetMoviesError (page 221) and GetMoviesStickyError (page 222). See Error Codes.

Discussion

Don't use SetTimeBaseMasterClock (page 304) to assign a clock component to a movie.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

Movies.h

SetMovieMasterTimeBase

Assigns a master time base to a movie.

```
void SetMovieMasterTimeBase (
   Movie theMovie,
   TimeBase tb,
   const TimeRecord *slaveZero
);
```

Parameters

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as NewMovie (page 259), NewMovieFromFile (page 1398), and NewMovieFromHandle (page 1400).

tb

The master time base to be assigned to this movie. Your application obtains this time base identifier from NewTimeBase (page 261).

slaveZero

A pointer to the time, in the time scale of the master time base, that corresponds to a 0 time value for the movie. This parameter allows you to set an offset between the movie and the master time base. Set this parameter to NIL if there is no offset.

Return Value

You can access error returns from this function through GetMoviesError (page 221) and GetMoviesStickyError (page 222). See Error Codes.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Show Movie

Declared In

Movies.h

SetMovieMatrix

Sets a movie's transformation matrix.

```
void SetMovieMatrix (
   Movie theMovie,
   const MatrixRecord *matrix
);
```

Parameters

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as NewMovie (page 259), NewMovieFromFile (page 1398), and NewMovieFromHandle (page 1400).

matrix

A pointer to the MatrixRecord structure for the movie. If you set this parameter to NIL, the Movie Toolbox uses the identity matrix.

Return Value

You can access error returns from this function through GetMoviesError (page 221) and GetMoviesStickyError (page 222). See Error Codes.

Discussion

The Movie Toolbox uses a movie's matrix to map a movie from its display coordinate system to its graphics world.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

OpenGLMovieQT

QTCarbonShell

vrmovies

vrscript

vrscript.win

Declared In
`Movies.h`

SetMoviePosterTime

Sets the poster time for the movie.

```
void SetMoviePosterTime (
   Movie theMovie,
   TimeValue posterTime
);
```

Parameters

`theMovie`
: The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

`posterTime`
: The starting time for the movie frame that contains the poster image, expressed in the movie's time coordinate system.

Return Value
You can access error returns from this function through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). See `Error Codes`.

Discussion
Since a movie poster is a still frame, it is defined by a point in time within the movie.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
bMoviePalette

bMoviePaletteCocoa

qtinfo

qtinfo.win

vrmakeobject

Declared In
`Movies.h`

SetMoviePreferredRate

Specifies a movie's default playback rate.

```
void SetMoviePreferredRate (
   Movie theMovie,
   Fixed rate
);
```

Parameters

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as NewMovie (page 259), NewMovieFromFile (page 1398), and NewMovieFromHandle (page 1400).

rate

The new movie rate as a 32-bit, fixed-point number. Positive integers indicate forward rates and negative integers indicate reverse rates.

Return Value

You can access error returns from this function through GetMoviesError (page 221) and GetMoviesStickyError (page 222). See Error Codes.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

Movies.h

SetMoviePreferredVolume

Sets a movie's preferred volume setting.

```
void SetMoviePreferredVolume (
   Movie theMovie,
   short volume
);
```

Parameters

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as NewMovie (page 259), NewMovieFromFile (page 1398), and NewMovieFromHandle (page 1400).

volume

The preferred volume setting of the movie. The volume parameter must contain a 16-bit, fixed-point number that contains the movie's default volume. The high-order 8 bits contain the integer part of the value; the low-order 8 bits contain the fractional part. Volume values range from -1.0 to 1.0. Negative values play no sound but preserve the absolute value of the volume setting. You may find the constants shown below useful. See these constants:

Return Value

You can access error returns from this function through GetMoviesError (page 221) and GetMoviesStickyError (page 222). See Error Codes.

Discussion

A movie's tracks may have their own volume settings. Use SetTrackVolume (page 1657) to set the volume of an individual track. A track's volume is scaled by the movie's volume to produce the track's final volume.

Special Considerations

After calling this function you must save the changes it has made, for example by updating or flattening the movie.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

MakeEffectMovie

qteffects.win

qtgraphics.win

QTMusicToo

vrbackbuffer.win

Declared In

`Movies.h`

SetMoviePreviewMode

Places a movie into and out of preview mode.

```
void SetMoviePreviewMode (
   Movie theMovie,
   Boolean usePreview
);
```

Parameters

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

usePreview

The movie's mode. Set this parameter to TRUE to place the movie into preview mode. Set this parameter to FALSE to place the movie into normal playback mode.

Return Value

You can access error returns from this function through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). See `Error Codes`.

Discussion

When a movie is in preview mode, only those tracks identified as preview tracks are serviced. You specify how a track is used by calling `SetTrackUsage` (page 1656).

When you place a movie into preview mode, the Movie Toolbox sets the active movie segment to be the preview segment of the movie. When you take a movie out of preview mode and place it back in normal playback mode, the toolbox sets the active movie segment to be the entire movie. For information about working with active movie segments, see `PrerollMovie` (page 264).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

SetMoviePreviewTime

Defines the starting time and duration of the movie's preview.

```
void SetMoviePreviewTime (
   Movie theMovie,
   TimeValue previewTime,
   TimeValue previewDuration
);
```

Parameters

theMovie
The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

previewTime
A time value that specifies the preview's starting time.

previewDuration
A time value that specifies the preview's duration.

Return Value
You can access error returns from this function through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qtinfo
qtinfo.win
vrmakeobject
vrmakeobject.win

Declared In
`Movies.h`

SetMovieRate

Sets a movie's playback rate.

```
void SetMovieRate (
   Movie theMovie,
   Fixed rate
);
```

Parameters

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as NewMovie (page 259), NewMovieFromFile (page 1398), and NewMovieFromHandle (page 1400).

rate

The new movie rate as a 32-bit, fixed-point number. Positive integers indicate forward rates and negative integers indicate reverse rates. This value immediately changes the rate at which the movie is playing. A value of 1 starts the movie playing at normal speed, a value of 2 causes the movie to play at double speed, -2 starts the movie playing backward at double speed, and so on. A value of 0 stops the movie.

Return Value

You can access error returns from this function through GetMoviesError (page 221) and GetMoviesStickyError (page 222). See Error Codes.

Discussion

Use this function to change the speed at which a movie is playing. You do not normally use this function to start and stop movies; use the higher level functions StartMovie (page 332) and StopMovie (page 333) instead. If you start a movie using this function, you should call PrePrerollMovie (page 263) and PrerollMovie (page 264) first, to set up any network connections, buffers, and data structures necessary to play the movie.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

AddFrameToMovie

OpenGLMovieQT

QTCarbonCoreImage101

Show Movie

vrscript.win

Declared In

`Movies.h`

SetMovieSelection

Sets a movie's current selection.

```
void SetMovieSelection (
   Movie theMovie,
   TimeValue selectionTime,
   TimeValue selectionDuration
);
```

Parameters

theMovie
: The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

selectionTime
: A time value specifying the starting point of the current selection.

selectionDuration
: A time value that specifies the duration of the current selection.

Return Value
You can access error returns from this function through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qtinfo
qtinfo.win
SlideShowImporter
SlideShowImporter.win

Declared In
`Movies.h`

SetMoviesErrorProc

Performs custom error notification.

```
void SetMoviesErrorProc (
   MoviesErrorUPP errProc,
   long refcon
);
```

Parameters

errProc
: A `MoviesErrorProc` callback.

refcon
: A reference constant value. The Movie Toolbox passes this reference constant to your `MoviesErrorProc` callback each time it calls it. Use this parameter to point to a data structure containing any information your callback needs.

Return Value
You can access error returns from this function through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). See `Error Codes`.

Discussion
Your application must identify its custom error-notification function to the Movie Toolbox. Once you have identified an error-notification function, the Movie Toolbox calls your function each time the current error value is to be set to a nonzero value. The Movie Toolbox calls your error-notification function only in response to errors generated by the Movie Toolbox.

Special Considerations

Error-notification functions can be especially useful when you are debugging your program. The Movie Toolbox manages the sticky error value.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

SetMovieTime

Changes a movie's current time.

```
void SetMovieTime (
   Movie theMovie,
   const TimeRecord *newtime
);
```

Parameters

`theMovie`
The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

`newtime`
A pointer to a `TimeRecord` structure containing the new time.

Return Value
You can access error returns from this function through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). See `Error Codes`.

Discussion
The Movie Toolbox saves the movie's current time when you save the movie.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTPixelBufferVCToCGImage

Declared In

`Movies.h`

SetMovieTimeScale

Establishes a movie's time scale.

```
ComponentResult ADD_MEDIA_BASENAME() SetMovieTimeScale
```

Parameters

`theMovie`

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

`timeScale`

The movie's new time scale.

Return Value

You can access error returns from this function through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

MakeEffectMovie

makeeffectslideshow

makeeffectslideshow.win

vrmakepano

vrmakepano.win

Declared In

`Movies.h`

SetMovieTimeValue

Sets a movie's time value.

```
void SetMovieTimeValue (
   Movie theMovie,
   TimeValue newtime
);
```

Parameters

`theMovie`

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

`newtime`

The new time value. You must ensure that the time value is in the movie's time scale.

Return Value
You can access error returns from this function through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CompressMovies
DigitizerShell
DragAndDrop Shell
MovieGWorlds
QT Internals

Declared In
`Movies.h`

SetMovieVideoOutput

Indicates to the ICM the video output component being used with a given movie.

```
void SetMovieVideoOutput (
   Movie theMovie,
   ComponentInstance vout
);
```

Parameters

`theMovie`
The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

`vout`
The video output component. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`. Call the function and pass `NIL` in this parameter as soon as the video output component is no longer in use.

Discussion
As soon as you turn on the echo port on any video output component, you should make this call so the ICM keeps track of the video output in use.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

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` (page 302) succeeds, it will retain the `QTVisualContext` object for its own use. If `visualContext` is NULL, the movie will not render any visual media. `SetMovieVisualContext` (page 302) will fail if a different movie is already using the visual context, so you should first `disassociate` the other movie by calling `SetMovieVisualContext` (page 302) with a NULL `visualContext`.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

QTCoreImage101

QTCoreVideo102

QTCoreVideo103

QTCoreVideo201

QTCoreVideo301

Declared In

`Movies.h`

SetMovieVolume

Sets a movie's current volume but does not store the setting in the movie.

```
void SetMovieVolume (
   Movie theMovie,
   short volume
);
```

Parameters

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

volume
The current volume setting of the movie represented as a 16-bit, fixed-point number. The high-order 8 bits contain the integer part of the value; the low-order 8 bits contain the fractional part. Volume values range from -1.0 to 1.0. Negative values play no sound but preserve the absolute value of the volume setting. You can use the constants shown below. See these constants:

Return Value
You can access error returns from this function through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). See `Error Codes`.

Discussion
The setting made by this function is not persistent. To store a volume setting in the movie, call `SetMoviePreferredVolume` (page 294).

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
MovieBrowser

QTMusicToo

SimpleCocoaMovie

vrscript

vrscript.win

Declared In
`Movies.h`

SetTimeBaseFlags

Sets the contents of the control flags of a time base.

```
void SetTimeBaseFlags (
   TimeBase tb,
   long timeBaseFlags
);
```

Parameters

tb
The time base for this operation. Your application obtains this time base identifier from `NewTimeBase` (page 261).

timeBaseFlags
The control flags for this time base (see below). You may set only one flag to 1. Be sure to set unused flags to 0. See these constants:
`loopTimeBase`
`palindromeLoopTimeBase`

Return Value
You can access error returns from this function through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
MakeEffectMovie

vrmovies

vrmovies.win

vrscript

vrscript.win

Declared In
`Movies.h`

SetTimeBaseMasterClock

Assigns a clock component to a time base.

```
void SetTimeBaseMasterClock (
   TimeBase slave,
   Component clockMeister,
   const TimeRecord *slaveZero
);
```

Parameters

slave
: The time base for this operation. Your application obtains this time base identifier from `NewTimeBase` (page 261).

clockMeister
: The clock component to be assigned to this time base. Your application can obtain this component identifier from `FindNextComponent`.

slaveZero
: A pointer to the time, in the clock's time scale, that corresponds to a 0 time value for the slave time base. This parameter allows you to set an offset between the time base and the clock component. Set this parameter to `NIL` if there is no offset.

Return Value
You can access error returns from this function through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). See `Error Codes`.

Discussion
A time base derives its time from either a clock component or from another time base. Don't use this function to assign a clock to a movie's time base.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code

BrideOfMungGrab

QTMusicToo

Declared In

`Movies.h`

SetTimeBaseMasterTimeBase

Assigns a master time base to a time base.

```
void SetTimeBaseMasterTimeBase (
   TimeBase slave,
   TimeBase master,
   const TimeRecord *slaveZero
);
```

Parameters

`slave`

The time base for this operation. Your application obtains this time base identifier from `NewTimeBase` (page 261).

`master`

The master time base to be assigned to this time base. Your application obtains this time base identifier from `NewTimeBase` (page 261).

`slaveZero`

A pointer to the time, in the time scale of the master time base, that corresponds to a 0 time value for the slave time scale. This parameter allows you to set an offset between the time base and the master time base. Set this parameter to `NIL` if there is no offset.

Return Value

You can access error returns from this function through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). See `Error Codes`.

Discussion

A time base derives its time from either a clock component or another time base. Don't use this function to assign a master time base to a movie's time base.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

LiveVideoMixer2

LiveVideoMixer3

QTMusicToo

Declared In

`Movies.h`

SetTimeBaseOffsetTimeBase

Attaches an offset time base to another time base.

```
OSErr SetTimeBaseOffsetTimeBase (
   TimeBase tb,
   TimeBase offsettb,
   const TimeRecord *offsetZero
);
```

Parameters

masterOffsetTimeBase

The time base to which the offset time base is to be attached. A `NIL` value can be passed when the offset time base has already be set but a new offset value is needed.

offsetTimeBase

The offset time base to be attached.

offsetZero

A pointer to a `TimeRecord` value set to the offset between the master time base and the offset time base. Passing a negative value means the offset time base will start sooner.

Return Value

See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 6.4.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`Movies.h`

SetTimeBaseRate

Sets the rate of a time base.

```
void SetTimeBaseRate (
   TimeBase tb,
   Fixed r
);
```

Parameters

tb

The time base for this operation. Your application obtains this time base identifier from `NewTimeBase` (page 261).

r

The rate of the time base.

Return Value

You can access error returns from this function through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). See `Error Codes`.

Discussion

Rates may be set to negative values. Negative rates cause time to move backward for the time base.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qteffects.win

QTMusicToo

qtshoweffect

VideoProcessing

vrscript.win

Declared In
`Movies.h`

SetTimeBaseStartTime

Sets the start time of a time base.

```
void SetTimeBaseStartTime (
   TimeBase tb,
   const TimeRecord *tr
);
```

Parameters

`tb`
: The time base for this operation. Your application obtains this time base identifier from `NewTimeBase` (page 261).

`tr`
: A pointer to a `TimeRecord` structure that contains the start time value.

Return Value
You can access error returns from this function through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). See `Error Codes`.

Discussion
The start time defines the time base's minimum time value. You must specify the new start time in a time structure.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

SetTimeBaseStopTime

Sets the stop time of a time base.

```
void SetTimeBaseStopTime (
   TimeBase tb,
   const TimeRecord *tr
);
```

Parameters

tb

The time base for this operation. Your application obtains this time base identifier from NewTimeBase (page 261).

tr

A pointer to a TimeRecord structure that contains the stop time value.

Return Value

You can access error returns from this function through GetMoviesError (page 221) and GetMoviesStickyError (page 222). See Error Codes.

Discussion

The stop time defines the time base's maximum time value. You must specify the new stop time in a time structure.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

Movies.h

SetTimeBaseTime

Sets the current time of a time base.

```
void SetTimeBaseTime (
   TimeBase tb,
   const TimeRecord *tr
);
```

Parameters

tb

The time base for this operation. Your application obtains this time base identifier from NewTimeBase (page 261).

tr

A pointer to a TimeRecord structure that contains the current time value.

Return Value

You can access error returns from this function through GetMoviesError (page 221) and GetMoviesStickyError (page 222). See Error Codes.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

SetTimeBaseValue

Sets the current time of a time base.

```
void SetTimeBaseValue (
   TimeBase tb,
   TimeValue t,
   TimeScale s
);
```

Parameters

tb

The time base for this operation. Your application obtains this time base identifier from `NewTimeBase` (page 261).

t

The new time value.

s

The time scale of the new time value.

Return Value
You can access error returns from this function through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qteffects.win

qtshoweffect

Show Movie

VideoProcessing

vrscript.win

Declared In
`Movies.h`

SetTimeBaseZero

Changes the offset from a time base to either its master time base or its clock component.

```
void SetTimeBaseZero (
   TimeBase tb,
   TimeRecord *zero
);
```

Parameters

tb

The time base for this operation. Your application obtains this time base identifier from NewTimeBase (page 261).

zero

A pointer to the time that corresponds to a 0 time value for the slave time scale. This parameter allows you to set an offset between the time base and its time source. Set this parameter to NIL if there is no offset.

Return Value

You can access error returns from this function through GetMoviesError (page 221) and GetMoviesStickyError (page 222). See Error Codes.

Discussion

You establish the initial offset when you assign the time base to its time source.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

Movies.h

SetTrackClipRgn

Sets the clipping region of a track.

```
void SetTrackClipRgn (
   Track theTrack,
   RgnHandle theClip
);
```

Parameters

theTrack

The track for this operation. Your application obtains this track identifier from such functions as NewMovieTrack (page 1628) and GetMovieTrack (page 1601).

theClip

A handle to the track's clipping region. The Movie Toolbox makes a copy of this region. Your application must dispose of the region referred to by this parameter when you are done with it. Set this parameter to NIL to disable clipping for the track.

Return Value

You can access error returns from this function through GetMoviesError (page 221) and GetMoviesStickyError (page 222). See Error Codes.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
AlwaysPreview

Declared In
`Movies.h`

SetTrackGWorld

Forces a track to draw into a particular graphics world, which may be different from that of the movie.

```
void SetTrackGWorld (
   Track theTrack,
   CGrafPtr port,
   GDHandle gdh,
   TrackTransferUPP proc,
   long refCon
);
```

Parameters

theTrack
The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

port
Points to the graphics port structure or graphics world to which to draw the `track`. Set this parameter to `NIL` to use the movie's graphics port.

gdh
A handle to the movie's graphics device structure. Set this parameter to `NIL` to use the current device. If the `port` parameter specifies a graphics world, set this parameter to `NIL` to use that graphics world's graphics device.

proc
A pointer to your `TrackTransferProc` callback. Set this parameter to `NIL` if you want to remove your callback.

refCon
A value to pass to your `TrackTransferProc` callback. Use this parameter to point to a data structure containing any information your callback needs.

Return Value
You can access error returns from this function through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). See `Error Codes`.

Discussion
After this function draws a track, it calls your transfer callback to copy the track to the actual movie graphics world. When your transfer callback is called, the current graphics world is set to the correct destination. You can also install a transfer callback and set the graphics world to `NIL`. In this case, the function calls your callback only as a notification that the track has been drawn; no transfer needs to take place.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
MovieGWorlds

Declared In
`Movies.h`

SetTrackMatte

Sets a track's matte.

```
void SetTrackMatte (
   Track theTrack,
   PixMapHandle theMatte
);
```

Parameters

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

theMatte

A handle to the matte. The Movie Toolbox makes a copy of the matte, including its `ColorTable` structure and pixels. Consequently, your application must dispose of the matte when you are done with it. Set this parameter to `NIL` to remove the track's matte.

Return Value
You can access error returns from this function through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). See `Error Codes`.

Discussion
This matte defines which of the track's pixels are displayed in a movie. You must specify the matte in a `PixMap` structure. The Movie Toolbox displays the weighted average of the track and its destination based on the corresponding pixel in the matte.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Inside Mac Movie TB Code

Declared In
`Movies.h`

ShowMoviePoster

Displays a movie's poster.

```
void ShowMoviePoster (
   Movie theMovie
);
```

Parameters

theMovie

A movie identifier. Your application obtains this identifier from such functions as NewMovie (page 259), NewMovieFromFile (page 1398), and NewMovieFromHandle (page 1400).

Return Value

You can access error returns from this function through GetMoviesError (page 221) and GetMoviesStickyError (page 222). See Error Codes.

Discussion

The Movie Toolbox draws the movie poster once, in the movie's graphics world, using the movie's matrix and display clipping characteristics. You can access error returns from this function through GetMoviesError (page 221) and GetMoviesStickyError (page 222). See Error Codes.

Special Considerations

This function works on both active and inactive movies.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

SpriteMediaCountImages

Retrieves the number of images that currently exist in a sprite track.

```
ComponentResult SpriteMediaCountImages (
   MediaHandler mh,
   short *numImages
);
```

Parameters

mh

The sprite media handler for this operation.

numImages

A pointer to a short integer. On return, this integer contains the number of images for the sprite media's current time.

Return Value

You can access Movie Toolbox error returns through GetMoviesError (page 221) and GetMoviesStickyError (page 222), as well as in the function result. See Error Codes.

Discussion

This function determines the number of images that currently exist based on the key frame that is in effect.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

SpriteMediaCountSprites

Retrieves the number of sprites that currently exist in a sprite track.

```
ComponentResult SpriteMediaCountSprites (
   MediaHandler mh,
   short *numSprites
);
```

Parameters

mh

The sprite media handler for this operation.

numSprites

A pointer to a short integer. On return, this integer contains the number of sprites for the sprite media's current time.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
This function determines the number of sprites that currently exist based on the key frame that is in effect.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

SpriteMediaDisposeImage

Frees the memory allocated for a sprite image outside a movie and removes that image from the sprite track in which it appears.

```
ComponentResult SpriteMediaDisposeImage (
   MediaHandler mh,
   short imageIndex
);
```

Parameters

mh

The sprite media handler for this operation.

imageIndex
The index of a sprite image that was previously created by `SpriteMediaNewImage` (page 326). If you know only the image ID, you can convert it to the index by calling `SpriteMediaImageIDToIndex` (page 325).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
The image disposed of is no longer available to the sprite track, and the image index location remains empty for the duration of the current key sample.

Version Notes
Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`Movies.h`

SpriteMediaDisposeSprite

Disposes of memory allocated for a sprite.

```
ComponentResult SpriteMediaDisposeSprite (
   MediaHandler mh,
   QTAtomID spriteID
);
```

Parameters

mh
The sprite media handler for this operation.

spriteID
The ID of the sprite for this operation.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

SpriteMediaGetActionVariable

Returns the value of the sprite track variable with the specified ID.

```
ComponentResult SpriteMediaGetActionVariable (
   MediaHandler mh,
   QTAtomID variableID,
   float *value
);
```

Parameters

mh

The sprite media handler for this operation.

variableID

A variable ID of the sprite variable.

value

A pointer to a floating-point value. If the specified variable has never been set, the value is set to 0 and the error `cannotFindAtomErr` is returned.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

SpriteMediaGetActionVariableAsString

Undocumented

```
ComponentResult SpriteMediaGetActionVariableAsString (
   MediaHandler mh,
   QTAtomID variableID,
   Handle *theCString
);
```

Parameters

mh

The sprite media handler for this operation.

variableID

A variable ID of the sprite variable.

theCString

A pointer to a handle to a C string.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

SpriteMediaGetDisplayedSampleNumber

Retrieves the number of the sprite media sample that is currently being displayed.

```
ComponentResult SpriteMediaGetDisplayedSampleNumber (
   MediaHandler mh,
   long *sampleNum
);
```

Parameters

mh

The sprite media handler for this operation.

sampleNum

A pointer to a long integer. On return, this integer contains the number of the sample that is currently being displayed.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

SpriteMediaGetImageName

Returns the name of the image with the specified index from the current key frame sample.

```
ComponentResult SpriteMediaGetImageName (
   MediaHandler mh,
   short imageIndex,
   Str255 imageName
);
```

Parameters

mh

The sprite media handler for this operation.

imageIndex

The index of the image whose image name is to be retrieved. This value must be between 1 and the number of available images. You can determine how many images are available by calling `SpriteMediaCountImages` (page 313).

imageName
Returns a Pascal string with the image name of the image, or an empty string if the image is unnamed.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

SpriteMediaGetIndImageDescription

Retrieves an image description for a specified image in a sprite track.

```
ComponentResult SpriteMediaGetIndImageDescription (
   MediaHandler mh,
   short imageIndex,
   ImageDescriptionHandle imageDescription
);
```

Parameters

mh
The sprite media handler for this operation.

imageIndex
The index of the image whose image description is to be retrieved. This value must be between 1 and the number of available images. You can determine how many images are available by calling `SpriteMediaCountImages` (page 313).

imageDescription
Specifies an image description handle. On return, this handle contains the `ImageDescription` structure that describes the specified image. This handle must be unlocked; the function resizes the handle if necessary.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

SpriteMediaGetIndImageProperty

Returns a property value for a sprite image specified by an index.

```
ComponentResult SpriteMediaGetIndImageProperty (
   MediaHandler mh,
   short imageIndex,
   long imagePropertyType,
   void *imagePropertyValue
);
```

Parameters

mh

The sprite media handler for this operation.

imageIndex

The index of the image whose property value is to be retrieved. This value must be between 1 and the number of available images. You can determine how many images are available by calling `SpriteMediaCountImages` (page 313).

imagePropertyType

The property whose value should be retrieved (see below). See these constants:

`kSpritePropertyMatrix`

`kSpritePropertyImageDescription`

`kSpritePropertyImageDataPtr`

`kSpritePropertyVisible`

`kSpritePropertyLayer`

`kSpritePropertyGraphicsMode`

imagePropertyValue

A pointer to a variable that will hold the selected property value on return.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

SpriteMediaGetProperty

Gets a sprite property; superseded by SpriteMediaGetSpriteProperty.

```
ComponentResult SpriteMediaGetProperty (
   MediaHandler mh,
   short spriteIndex,
   long propertyType,
   void *propertyValue
);
```

Parameters

mh

The sprite media handler for this operation.

spriteIndex

The index of the sprite for this operation.

propertyType

The property whose value should be retrieved (see below). See these constants:

`kSpritePropertyMatrix`

`kSpritePropertyImageDescription`

`kSpritePropertyImageDataPtr`

`kSpritePropertyVisible`

`kSpritePropertyLayer`

`kSpritePropertyGraphicsMode`

propertyValue

A pointer to a variable that will hold the selected property value on return.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

SpriteMediaGetSpriteActionsForQTEvent

Gets the sprite action atom for an event.

```
ComponentResult SpriteMediaGetSpriteActionsForQTEvent (
   MediaHandler mh,
   QTEventRecordPtr event,
   QTAtomID spriteID,
   QTAtomContainer *container,
   QTAtom *atom
);
```

Parameters

mh

The sprite media handler for this operation.

event
A pointer to a `QTEventRecord` structure.

spriteID
The ID of the sprite for this operation.

container
A pointer to a QT atom container.

atom
A pointer to a QT atom in the container.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

SpriteMediaGetSpriteName

Returns the name of the sprite with the specified ID from the currently displayed sample.

```
ComponentResult SpriteMediaGetSpriteName (
   MediaHandler mh,
   QTAtomID spriteID,
   Str255 spriteName
);
```

Parameters

mh
The sprite media handler for this operation.

spriteID
The sprite ID of the sprite name.

spriteName
Returns a Pascal string with the name of the sprite or an empty string if the sprite is unnamed.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

SpriteMediaGetSpriteProperty

Retrieves the value of the specified sprite or sprite track property.

```
ComponentResult SpriteMediaGetSpriteProperty (
   MediaHandler mh,
   QTAtomID spriteID,
   long propertyType,
   void *propertyValue
);
```

Parameters

mh

The sprite media handler for this operation.

spriteID

The ID of the sprite for this operation. Pass `'Trck'` to return the properties of a whole sprite track.

propertyType

A constant (see below) that specifies the property whose value should be retrieved. See these constants:

- `kSpritePropertyMatrix`
- `kSpritePropertyImageDescription`
- `kSpritePropertyImageDataPtr`
- `kSpritePropertyVisible`
- `kSpritePropertyLayer`
- `kSpritePropertyGraphicsMode`
- `kSpriteTrackPropertyAllSpritesHitTestingMode`
- `kSpriteTrackPropertyPreferredDepthInterpretationMode`

propertyValue

On return, a pointer to the value of the `property`; the `data` type of that value depends on the property.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Function introduced in QuickTime 3 or earlier. The `kSpriteTrackPropertyAllSpritesHitTestingMode` and `kSpriteTrackPropertyPreferredDepthInterpretationMode` constants were added in QuickTime 6.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

MovieSprites

qtsprites.win

qtspritesplus.win

vrscript

vrscript.win

Declared In

`Movies.h`

SpriteMediaHitTestAllSprites

Determines whether any sprites are at a specified location.

```
ComponentResult SpriteMediaHitTestAllSprites (
   MediaHandler mh,
   long flags,
   Point loc,
   QTAtomID *spriteHitID
);
```

Parameters

mh

The sprite media handler for this operation.

flags

Specifies flags (see below) that control the hit testing operation. See these constants:

`spriteHitTestBounds`

`spriteHitTestImage`

`spriteHitTestInvisibleSprites`

`spriteHitTestIsClick`

`spriteHitTestLocInDisplayCoordinates`

loc

A point in the coordinate system of the sprite track's movie to test for the existence of a sprite.

spriteHitID

A pointer to a short integer. On return, this integer contains the ID of the frontmost sprite at the location specified by `loc`. If no sprite exists at the location, the function sets the `value` of this parameter to 0.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

You call this function to determine whether any sprites exist at a specified location in the coordinate system of a sprite track's movie. You can pass flags to this function to control the hit testing operation more precisely. For example, you may want the hit test operation to detect a sprite whose bounding box contains the specified location.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

MovieSprites

qtsprites

qtsprites.win

qtspritesplus.win

vrscript.win

Declared In

`Movies.h`

SpriteMediaHitTestOneSprite

Performs a hit testing operation on the sprite specified by a spriteID.

```
ComponentResult SpriteMediaHitTestOneSprite (
   MediaHandler mh,
   QTAtomID spriteID,
   long flags,
   Point loc,
   Boolean *wasHit
);
```

Parameters

mh

The sprite media handler for this operation.

spriteID

The sprite ID of the sprite.

flags

Flags (see below) that control the hit testing operation. See these constants:

`spriteHitTestBounds`
`spriteHitTestImage`
`spriteHitTestInvisibleSprites`
`spriteHitTestIsClick`
`spriteHitTestLocInDisplayCoordinates`

loc

A point to test for the existence of a sprite. The point should be defined in the local coordinates of the sprite track, unless the `spriteHitTestLocInDisplayCoordinates` flag is set.

wasHit

A pointer to a Boolean. If the sprite is hit, `wasHit` is set to TRUE; otherwise, it is set to FALSE.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

This routine allows you to hit test a sprite which is fully or partially covered by other sprites.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

SpriteMediaHitTestSprites

Undocumented

```
ComponentResult SpriteMediaHitTestSprites (
   MediaHandler mh,
   long flags,
   Point loc,
   short *spriteHitIndex
);
```

Parameters

mh

The sprite media handler for this operation.

flags

Flags (see below) that control the hit testing operation. See these constants:

`spriteHitTestBounds`

`spriteHitTestImage`

`spriteHitTestInvisibleSprites`

`spriteHitTestIsClick`

`spriteHitTestLocInDisplayCoordinates`

loc

A point to test for the existence of a sprite.

spriteHitIndex

Undocumented

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

SpriteMediaImageIDToIndex

Returns the index of an outside sprite image from the ID of that image.

```
ComponentResult SpriteMediaImageIDToIndex (
   MediaHandler mh,
   QTAtomID imageID,
   short *imageIndex
);
```

Parameters

mh

The sprite media handler for this operation.

imageID

The ID of a sprite image.

imageIndex
    On return, a pointer to the index of the image.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`Movies.h`

SpriteMediaImageIndexToID

Returns the ID of an outside sprite image from the index of that image.

```
ComponentResult SpriteMediaImageIndexToID (
   MediaHandler mh,
   short imageIndex,
   QTAtomID *imageID
);
```

Parameters

mh
    The sprite media handler for this operation.

imageIndex
    The index of a sprite image.

imageID
    On return, a pointer to the ID of the image.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`Movies.h`

SpriteMediaNewImage

Creates a new movie sprite image outside a movie.

```
ComponentResult SpriteMediaNewImage (
   MediaHandler mh,
   Handle dataRef,
   OSType dataRefType,
   QTAtomID desiredID
);
```

Parameters

mh

The sprite media handler for this operation.

dataRef

A pointer to a URL or an alias that references the image to be used as a sprite image.

dataRefType

A four character code for the type of the `dataRef` parameter. See `Component Identifiers`. For example, pass `URLDataHandlerSubType` if `dataRef` is a URL

desiredID

The desired ID identifier for the image. If the requested ID is in use, the call returns an error. If you pass 0 the function assigns the next sequential integer ID, which is usually the same as the next available index unless that ID has been previously assigned.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The newly created image can be used in a sprite track like any other sprite image. It can be referenced by the next available image index, equal to the number of images in the track before the call was made +1, or by the ID that was requested via the `desiredID` parameter.

Version Notes

Introduced in QuickTime 6.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`Movies.h`

SpriteMediaNewSprite

Creates a new sprite.

```
ComponentResult SpriteMediaNewSprite (
   MediaHandler mh,
   QTRuntimeSpriteDescPtr newSpriteDesc
);
```

Parameters

mh

The sprite media handler for this operation.

newSpriteDesc

A pointer to a `QTRuntimeSpriteDescStruct` structure.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

SpriteMediaSetActionVariable

Sets the value of a sprite track variable to a specified value.

```
ComponentResult SpriteMediaSetActionVariable (
   MediaHandler mh,
   QTAtomID variableID,
   const float *value
);
```

Parameters

mh

The sprite media handler for this operation.

variableID

A variable ID of the sprite name.

value

A pointer to a floating-point number. The value is passed by reference.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

This function is specific to sprite tracks using wired sprites.

Special Considerations

This function is specific to sprite tracks using wired sprites.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

SpriteMediaSetActionVariableToString

Undocumented

```
ComponentResult SpriteMediaSetActionVariableToString (
   MediaHandler mh,
   QTAtomID variableID,
   Ptr theCString
);
```

Parameters

mh

The sprite media handler for this operation.

variableID

A variable ID of the sprite variable.

theCString

A pointer to a C string.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

SpriteMediaSetProperty

Sets a sprite property; superseded by SpriteMediaSetSpriteProperty.

```
ComponentResult SpriteMediaSetProperty (
   MediaHandler mh,
   short spriteIndex,
   long propertyType,
   void *propertyValue
);
```

Parameters

mh

The sprite media handler for this operation.

spriteIndex

The index of the sprite for this operation.

propertyType

The property whose value should be set (see below). See these constants:

`kSpritePropertyMatrix`

`kSpritePropertyImageDescription`

`kSpritePropertyImageDataPtr`

`kSpritePropertyVisible`

`kSpritePropertyLayer`

`kSpritePropertyGraphicsMode`

propertyValue
A pointer to a variable that contains the new value of the selected property. The type of data you pass for this parameter depends on the `property` type.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

SpriteMediaSetSpriteProperty

Sets the specified property of a sprite or sprite track.

```
ComponentResult SpriteMediaSetSpriteProperty (
   MediaHandler mh,
   QTAtomID spriteID,
   long propertyType,
   void *propertyValue
);
```

Parameters

mh
The sprite media handler for this operation.

spriteID
The ID of the sprite for this operation. Pass `'Trck'` to set the properties of a whole sprite track.

propertyType
A constant (see below) that specifies the property whose value should be set. See these constants:
`kSpritePropertyMatrix`
`kSpritePropertyImageDescription`
`kSpritePropertyImageDataPtr`
`kSpritePropertyVisible`
`kSpritePropertyLayer`
`kSpritePropertyGraphicsMode`
`kSpriteTrackPropertyAllSpritesHitTestingMode`
`kSpriteTrackPropertyPreferredDepthInterpretationMode`

propertyValue
A pointer to the new value of the selected property. The type of data you pass for this parameter depends on the `property` type.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Function introduced in QuickTime 3 or earlier. The `kSpriteTrackPropertyAllSpritesHitTestingMode` and `kSpriteTrackPropertyPreferredDepthInterpretationMode` constants were added in QuickTime 6.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
MovieSprites

qtsprites.win

qtspritesplus.win

vrscript

vrscript.win

Declared In
`Movies.h`

SpriteMediaSpriteIDToIndex

Converts a sprite ID to the corresponding sprite index.

```
ComponentResult SpriteMediaSpriteIDToIndex (
   MediaHandler mh,
   QTAtomID spriteID,
   short *spriteIndex
);
```

Parameters

mh
The sprite media handler for this operation.

spriteID
The ID of the sprite for this operation.

spriteIndex
On return, a pointer to the index of the sprite.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

SpriteMediaSpriteIndexToID

Returns the ID of a sprite specified by a sprite index.

```
ComponentResult SpriteMediaSpriteIndexToID (
   MediaHandler mh,
   short spriteIndex,
   QTAtomID *spriteID
);
```

Parameters

mh

The sprite media handler for this operation.

spriteIndex

The index of the sprite for this operation.

spriteID

A pointer to the sprite ID corresponding to the sprite index. If a sprite with the specified index does not exist, the error `paramErr` is returned.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

StartMovie

Starts the movie playing from the current movie time.

```
void StartMovie (
   Movie theMovie
);
```

Parameters

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

Return Value

You can access error returns from this function through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). See `Error Codes`.

Discussion

You are not required to call this function to start a movie. It is included in the QuickTime API for convenience. Before playing the movie, the Movie Toolbox makes the movie active, prerolls the movie, and sets the movie to its preferred playback rate. You can use `SetMoviePreferredRate` (page 293) to change this setting.

Special Considerations

A movie's current time is saved when a movie is stored in a movie file. Therefore, your application should appropriately position a movie before playing the movie. Use `GoToBeginningOfMovie` (page 240) to set a movie to play from its start.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CarbonQTGraphicImport

Graphic Import-Export

MovieGWorlds

OpenGLCompositorLab

vrscript.win

Declared In
`Movies.h`

StopMovie

Stops the playback of a movie.

```
void StopMovie (
   Movie theMovie
);
```

Parameters
theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CarbonQTGraphicImport

Graphic Import-Export

vrmovies

vrscript

vrscript.win

Declared In
`Movies.h`

SubtractTime

Subtracts one time from another.

```
void SubtractTime (
   TimeRecord *dst,
   const TimeRecord *src
);
```

Parameters

dst

A pointer to a TimeRecord structure. This time structure contains one of the operands for the subtraction. This function returns the result of the subtraction into this time structure as a duration.

src

A pointer to a TimeRecord structure. The Movie Toolbox subtracts this value from the time or duration specified by the dst parameter.

Return Value

You can access error returns from this function through GetMoviesError (page 221) and GetMoviesStickyError (page 222). See Error Codes.

Discussion

If the two times are relative to different time scales or time bases, this function converts the times as appropriate to yield reasonable results. However, the time bases for both time values must rely on the same time source.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

Movies.h

TextMediaAddHiliteSample

Provides dynamic highlighting of text.

```
ComponentResult TextMediaAddHiliteSample (
   MediaHandler mh,
   short hiliteStart,
   short hiliteEnd,
   RGBColor *rgbHiliteColor,
   TimeValue duration,
   TimeValue *sampleTime
);
```

Parameters

mh

The media handler for the text media obtained by GetMediaHandler (page 1577).

hiliteStart

Indicates the beginning of the text to be highlighted.

hiliteEnd

Indicates the ending of the text to be highlighted. If the value of the hiliteStart parameter equals that of the hiliteEnd parameter, then no text is highlighted (that is, highlighting is turned off for the duration of the specified sample).

rgbHiliteColor
A pointer to the `RGBColor` structure that defines the color for highlighting. If this parameter is not `NIL`, then the specified color is used when highlighting the text indicated by the `hiliteStart` and `hiliteEnd` parameters. Otherwise, the default system highlight color is used.

duration
Specifies how long the text sample should last. This duration is expressed in the media's time base.

sampleTime
A pointer to a time value. The actual media time at which the sample was added is returned here.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

TextMediaAddTESample

Specifies a TextEdit handle to be added to a specified media.

```
ComponentResult TextMediaAddTESample (
   MediaHandler mh,
   TEHandle hTE,
   RGBColor *backColor,
   short textJustification,
   Rect *textBox,
   long displayFlags,
   TimeValue scrollDelay,
   short hiliteStart,
   short hiliteEnd,
   RGBColor *rgbHiliteColor,
   TimeValue duration,
   TimeValue *sampleTime
);
```

Parameters

mh
The media handler for the text media obtained by `GetMediaHandler` (page 1577).

hTE
A handle to a `TERec` structure.

backColor
A pointer to an `RGBColor` structure specifying the text background color. Passing `NIL` for this parameter in defaults to white.

textJustification
: Indicates the justification of the text (see below). See these constants:

textBox
: A pointer to a `Rect` structure that defines the box within which the text is to be displayed. The box is relative to the track bounds.

displayFlags
: Contains the text display flags (see below). See these constants:
  - `dfDontDisplay`
  - `dfDontAutoScale`
  - `dfClipToTextBox`
  - `dfShrinkTextBoxToFit`
  - `dfScrollIn`
  - `dfScrollOut`
  - `dfHorizScroll`
  - `dfReverseScroll`
  - `dfContinuousScroll`
  - `dfFlowHoriz`
  - `dfContinuousKaraoke`
  - `dfDropShadow`
  - `dfAntiAlias`
  - `dfKeyedText`
  - `dfInverseHilite`
  - `dfTextColorHilite`

scrollDelay
: Indicates the delay in scrolling associated with the setting of the `dfScrollIn` and `dfScrollOut` display flags. If the value of the `scrollDelay` parameter is greater than 0 and the `dfScrollIn` flag is set, the text pauses when it has scrolled all the way in for the amount of time specified by `scrollDelay`. If the `dfScrollOut` flag is set, the pause occurs first before the text scrolls out. If both these flags are set, the pause occurs at the midpoint between scrolling in and scrolling out.

hiliteStart
: The beginning of the text to be highlighted.

hiliteEnd
: The end of the text to be highlighted. If the `hiliteEnd` parameter is greater than the `hiliteStart` parameter, then the text is highlighted from the selection specified by `hiliteStart` to `hiliteEnd`. To specify additional highlighting, you can use `TextMediaAddHiliteSample` (page 334).

rgbHiliteColor
: Contains a pointer to an `RGBColor` structure that defines the color for highlighting. If this parameter is not `NIL`, then the specified color is used when highlighting the text indicated by the `hiliteStart` and `hiliteEnd` parameters. Otherwise, the default system highlight color is used.

duration
: A time value that specifies how long the text sample should last. This duration is expressed in the media's time base.

sampleTime
: Contains a pointer to a time value. The actual media time at which the sample was added is returned here.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Special Considerations

Be sure to turn on the `dfDropShadow` display flag after you call this function.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

TextMediaAddTextSample

Adds a single block of styled text to an existing media.

```
ComponentResult TextMediaAddTextSample (
   MediaHandler mh,
   Ptr text,
   unsigned long size,
   short fontNumber,
   short fontSize,
   Style textFace,
   RGBColor *textColor,
   RGBColor *backColor,
   short textJustification,
   Rect *textBox,
   long displayFlags,
   TimeValue scrollDelay,
   short hiliteStart,
   short hiliteEnd,
   RGBColor *rgbHiliteColor,
   TimeValue duration,
   TimeValue *sampleTime
);
```

Parameters

`mh`
  The media handler for the text media obtained by `GetMediaHandler` (page 1577).

`text`
  A pointer to a block of text.

`size`
  Indicates the size of the text block, in bytes.

`fontNumber`
  The number for the font in which to display the text.

`fontSize`
  Indicates the size of the font.

textFace
Indicates the typeface or `style` of the text (that is, bold, italic, and so on).

textColor
A pointer to an `RGBColor` structure specifying the color of the text. Passing `NIL` for this parameter in defaults to black.

backColor
A pointer to an `RGBColor` structure specifying the text background color. Passing `NIL` for this parameter in defaults to white.

textJustification
Indicates the justification of the text (see below). See these constants:

textBox
A pointer to a `Rect` structure that defines the box within which the text is to be displayed. The box is relative to the track bounds.

displayFlags
Contains the text display flags (see below). See these constants:

`dfDontDisplay`
`dfDontAutoScale`
`dfClipToTextBox`
`dfShrinkTextBoxToFit`
`dfScrollIn`
`dfScrollOut`
`dfHorizScroll`
`dfReverseScroll`
`dfContinuousScroll`
`dfFlowHoriz`
`dfContinuousKaraoke`
`dfDropShadow`
`dfAntiAlias`
`dfKeyedText`
`dfInverseHilite`
`dfTextColorHilite`

scrollDelay
Indicates the delay in scrolling associated with the setting of the `dfScrollIn` and `dfScrollOut` display flags. If the value of the `scrollDelay` parameter is greater than 0 and the `dfScrollIn` flag is set, the text pauses when it has scrolled all the way in for the amount of time specified by `scrollDelay`. If the `dfScrollOut` flag is set, the pause occurs first before the text scrolls out. If both these flags are set, the pause occurs at the midpoint between scrolling in and scrolling out.

hiliteStart
The beginning of the text to be highlighted.

hiliteEnd
The end of the text to be highlighted. If the `hiliteEnd` parameter is greater than the `hiliteStart` parameter, then the text is highlighted from the selection specified by `hiliteStart` to `hiliteEnd`. To specify additional highlighting, you can use `TextMediaAddHiliteSample` (page 334).

rgbHiliteColor

Contains a pointer to an RGBColor structure that defines the color for highlighting. If this parameter is not NIL, then the specified color is used when highlighting the text indicated by the hiliteStart and hiliteEnd parameters. Otherwise, the default system highlight color is used.

duration

A time value that specifies how long the text sample should last. This duration is expressed in the media's time base.

sampleTime

Contains a pointer to a time value. The actual media time at which the sample was added is returned here.

Return Value

You can access Movie Toolbox error returns through GetMoviesError (page 221) and GetMoviesStickyError (page 222), as well as in the function result. See Error Codes.

Special Considerations

Be sure to turn on the dfDropShadow display flag after you call this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

addhtactions.win

qttext

qttext.win

qtwiredactions

qtwiredactions.win

Declared In

Movies.h

TextMediaDrawRaw

Undocumented

```
ComponentResult TextMediaDrawRaw (
   MediaHandler mh,
   GWorldPtr gw,
   GDHandle gd,
   void *data,
   long dataSize,
   TextDescriptionHandle tdh
);
```

Parameters

mh

The text media handler obtained by GetMediaHandler (page 1577).

gw

A pointer to a CGrafPort structure that defines a graphics world.

gd
: A handle to a graphics device.

data
: A pointer to the source data.

dataSize
: The size of the source data.

tdh
: A handle to a `TextDescription` structure.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

TextMediaFindNextText

Searches for text with a specified media handler starting at a given time.

```
ComponentResult TextMediaFindNextText (
   MediaHandler mh,
   Ptr text,
   long size,
   short findFlags,
   TimeValue startTime,
   TimeValue *foundTime,
   TimeValue *foundDuration,
   long *offset
);
```

Parameters

mh
: The media handler for the text media obtained by `GetMediaHandler` (page 1577).

text
: Points to the text to be found.

size
: The length of the text to be found.

findFlags
: Flags (see below) that determine the conditions of the search. See these constants:
   - `findTextEdgeOK`
   - `findTextCaseSensitive`
   - `findTextReverseSearch`
   - `findTextWrapAround`
   - `findTextUseOffset`

startTime

Indicates the time (expressed in the movie time scale) at which to begin the search.

foundTime

A pointer to the movie time at which the text sample is found if the search is successful. Otherwise, it returns -1.

foundDuration

A pointer to the duration of the sample (in the movie time scale) that is found if the search is successful.

offset

A pointer to the offset of the found text from the beginning of the text portion of the sample.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qttext

qttext.win

Declared In

`Movies.h`

TextMediaGetTextProperty

Sets properties of a text media.

```
ComponentResult TextMediaGetTextProperty (
   MediaHandler mh,
   TimeValue atMediaTime,
   long propertyType,
   void *data,
   long dataSize
);
```

Parameters

mh

The text media handler obtained by `GetMediaHandler` (page 1577).

atMediaTime

The media time of the text.

propertyType

A constant (see below) that identifies the text property to be set. See these constants:

- `kTextTextHandle`
- `kTextTextPtr`
- `kTextTEStyle`
- `kTextBackColor`
- `kTextForeColor`
- `kTextFace`
- `kTextFont`
- `kTextSize`
- `kTextAlignment`
- `kTextHilite`
- `kTextDropShadow`
- `kTextDisplayFlags`
- `kTextScroll`

data

A pointer to text data.

dataSize

The size of the data.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

TextMediaHiliteTextSample

Specifies selected text to be highlighted for a given text media handler.

```
ComponentResult TextMediaHiliteTextSample (
   MediaHandler mh,
   TimeValue sampleTime,
   short hiliteStart,
   short hiliteEnd,
   RGBColor *rgbHiliteColor
);
```

Parameters

mh

The text media handler obtained by `GetMediaHandler` (page 1577).

sampleTime

The starting time in the sample.

hiliteStart
The beginning of the text to be highlighted.

hiliteEnd
The end of the text to be highlighted. If the `hiliteEnd` parameter is greater than the `hiliteStart` parameter, then the text is highlighted from the selection specified by `hiliteStart` to `hiliteEnd`.

rgbHiliteColor
Contains a pointer to an `RGBColor` structure that defines the color for highlighting. If this parameter is not `NIL`, then the specified color is used when highlighting the text indicated by the `hiliteStart` and `hiliteEnd` parameters. Otherwise, the default system highlight color is used.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qttext

qttext.win

Declared In
`Movies.h`

TextMediaRawIdle

Undocumented

```
ComponentResult TextMediaRawIdle (
   MediaHandler mh,
   GWorldPtr gw,
   GDHandle gd,
   TimeValue sampleTime,
   long flagsIn,
   long *flagsOut
);
```

Parameters

mh
The text media handler obtained by `GetMediaHandler` (page 1577).

gw
A pointer to a `CGrafPort` structure that defines a graphics world.

gd
A handle to a graphics device.

sampleTime
Undocumented

flagsIn
Undocumented

flagsOut

Undocumented

Return Value

You can access Movie Toolbox error returns through GetMoviesError (page 221) and GetMoviesStickyError (page 222), as well as in the function result. See Error Codes.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

Movies.h

TextMediaRawSetup

Undocumented

```
ComponentResult TextMediaRawSetup (
   MediaHandler mh,
   GWorldPtr gw,
   GDHandle gd,
   void *data,
   long dataSize,
   TextDescriptionHandle tdh,
   TimeValue sampleDuration
);
```

Parameters

mh

The text media handler obtained by GetMediaHandler (page 1577).

gw

A pointer to a CGrafPort structure that defines a graphics world.

gd

A handle to a graphics device.

data

A pointer to data.

dataSize

The size of the data.

tdh

A handle to a TextDescription structure.

sampleDuration

Undocumented

Return Value

You can access Movie Toolbox error returns through GetMoviesError (page 221) and GetMoviesStickyError (page 222), as well as in the function result. See Error Codes.

Version Notes

Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

TextMediaSetTextProc

Specifies a custom function to be called whenever a text sample is displayed in a movie.

```
ComponentResult TextMediaSetTextProc (
   MediaHandler mh,
   TextMediaUPP TextProc,
   long refcon
);
```

Parameters

mh
The text media handler obtained by `GetMediaHandler` (page 1577).

TextProc
A Universal Procedure Pointer that points to a `TextMediaProc` callback.

refcon
Indicates a reference constant that will be passed to your callback. Use this parameter to point to a data structure containing any information your function needs. Set this parameter to 0 if you don't need it.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qttext
qttext.win

Declared In
`Movies.h`

TextMediaSetTextProperty

Sets properties of a text media.

```
ComponentResult TextMediaSetTextProperty (
   MediaHandler mh,
   TimeValue atMediaTime,
   long propertyType,
   void *data,
   long dataSize
);
```

Parameters

mh
The text media handler obtained by `GetMediaHandler` (page 1577).

atMediaTime
The media time of the text.

propertyType
A constant (see below) that identifies the text property to be set. See these constants:

`kTextTextHandle`

`kTextTextPtr`

`kTextTEStyle`

`kTextBackColor`

`kTextForeColor`

`kTextFace`

`kTextFont`

`kTextSize`

`kTextAlignment`

`kTextHilite`

`kTextDropShadow`

`kTextDisplayFlags`

`kTextScroll`

data
A pointer to text data.

dataSize
The size of the data.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

TextMediaSetTextSampleData

Sets values before calling TextMediaAddTextSample or TextMediaAddTESample.

```
ComponentResult TextMediaSetTextSampleData (
   MediaHandler mh,
   void *data,
   OSType dataType
);
```

Parameters

mh

A reference to the text media handler. You obtain this reference from GetMediaHandler (page 1577).

data

A pointer to the data, defined by the dataType parameter.

dataType

The type of data.

Return Value

You can access Movie Toolbox error returns through GetMoviesError (page 221) and GetMoviesStickyError (page 222), as well as in the function result. See Error Codes.

Discussion

The following code sample demonstrates how to use this function:

```
// TextMediaSetTextSampleData coding example
short           trans =127;
Point           dropOffset;
MediaHandler    mh;
dropOffset.h =dropOffset.v =4
TextMediaSetTextSampleData(mh,(void *)&dropOffset,dropShadowOffsetType);
TextMediaSetTextSampleData(mh,(void *)&trans,dropShadowTranslucencyType);
```

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

Movies.h

UpdateMovie

Ensures that the Movie Toolbox properly displays your movie after it has been uncovered.

```
OSErr UpdateMovie (
   Movie theMovie
);
```

Parameters

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as NewMovie (page 259), NewMovieFromFile (page 1398), and NewMovieFromHandle (page 1400).

Return Value

You can access Movie Toolbox error returns through GetMoviesError (page 221) and GetMoviesStickyError (page 222), as well as in the function result. See Error Codes.

Discussion

Your application should call this function during window updating. Don't call MoviesTask (page 257) at this time; you will observe better display behavior if you call it at the end of your update processing.

This function does not actually update the movie's graphics world. Rather, it invalidates the movie's display state so that the Movie Toolbox redraws the movie the next time you call MoviesTask. If you need to force a movie to be redrawn outside of a window update sequence, your application can call this function and then call MoviesTask to service the movie. The Movie Toolbox determines the portion of the screen to update by examining the graphics port's visible region.

The following code snippet uses this function in a Macintosh Window Manager update sequence:

```
// UpdateMovie coding example
#include <Events.h>
#include <ToolUtils.h>
#include "Movies.h"
void DoUpdate (WindowRef theWindow, Movie theMovie)
{
    BeginUpdate (theWindow);
    UpdateMovie (theMovie);
    EndUpdate (theWindow);
} /* DoUpdate */
```

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CompressMovies

DigitizerShell

DragAndDrop Shell

mfc.win

MovieGWorlds

Declared In

Movies.h

VideoMediaGetCodecParameter

Undocumented

```
ComponentResult VideoMediaGetCodecParameter (
   MediaHandler mh,
   CodecType cType,
   OSType parameterID,
   Handle outParameterData
);
```

Parameters

mh

A reference to a video media handler. You obtain this reference from GetMediaHandler (page 1577).

cType
A valid codec type constant; see `Codec Identifiers`.

parameterID
Undocumented

outParameterData
A handle to the returned data.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

VideoMediaGetStallCount

Undocumented

```
ComponentResult VideoMediaGetStallCount (
   MediaHandler mh,
   unsigned long *stalls
);
```

Parameters

mh
A reference to a video media handler. You obtain this reference from `GetMediaHandler` (page 1577).

stalls
The number of stalls.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

VideoMediaGetStatistics

Returns the play-back frame rate of a movie.

```
ComponentResult VideoMediaGetStatistics (
   MediaHandler mh
);
```

Parameters

mh

A reference to a video media handler. You obtain this reference from `GetMediaHandler` (page 1577).

Return Value

The average frame rate since the last time `VideoMediaResetStatistics` (page 350) was called. Because of sampling errors, the values returned from this function are accurate only after waiting at least one second after calling `VideoMediaResetStatistics`.

Discussion

This function can only be used on video or MPEG media handlers. Because not all QuickTime movies have a constant frame rate, the results of this call can be difficult to interpret correctly. For this reason, the results of this function should not be displayed in a place where a novice user is likely to see it.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

VideoMediaResetStatistics

Resets the video media handler's counters before using VideoMediaGetStatistics to determine the frame rate of a movie.

```
ComponentResult VideoMediaResetStatistics (
   MediaHandler mh
);
```

Parameters

mh

A reference to a video media handler. You obtain this reference from `GetMediaHandler` (page 1577).

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Special Considerations

This call can only be used on video or MPEG media handlers.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

VideoMediaSetCodecParameter

Undocumented

```
ComponentResult VideoMediaSetCodecParameter (
   MediaHandler mh,
   CodecType cType,
   OSType parameterID,
   long parameterChangeSeed,
   void *dataPtr,
   long dataSize
);
```

Parameters

mh

A reference to a video media handler. You obtain this reference from GetMediaHandler (page 1577).

cType

A valid codec type constant; see Codec Identifiers.

parameterID

Undocumented

parameterChangeSeed

Undocumented

dataPtr

A pointer to the data to be set.

dataSize

The size of the data.

Return Value

You can access Movie Toolbox error returns through GetMoviesError (page 221) and GetMoviesStickyError (page 222), as well as in the function result. See Error Codes.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

Movies.h

Callbacks

Data Types

ControlPtr

Represents a type used by the Movie Manager API.

```
typedef ControlRecord * ControlPtr;
```

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In
`Controls.h`

ControlRef

Represents a type used by the Movie Manager API.

```
typedef ControlPtr * ControlRef;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HIObject.h`

QTFloatSingle

Represents a type used by the Movie Manager API.

```
typedef Float32 QTFloatSingle;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

QTNewMoviePropertyElement

Stores a movie property for NewMovieFromProperties.

```
struct QTNewMoviePropertyElement {
QTPropertyClass       propClass;
QTPropertyID          propID;
ByteCount             propValueSize;
QTPropertyValuePtr    propValueAddress;
OSStatus              propStatus;
};
```

Fields
`propClass`
Discussion
A four-character code designating the class of a movie property. See `New Movie Property Codes`.

`propID`
Discussion
The ID of the property.

`propValueSize`

Discussion
The size in bytes of the property passed in `propValueAddress`.

`propValueAddress`

Discussion
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`

Discussion
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 260).

Discussion
When you call `NewMovieFromProperties` (page 260), 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.

Related Functions
Associated function: `NewMovieFromProperties` (page 260)

Declared In
`Movies.h`

QTRuntimeSpriteDescPtr

Represents a type used by the Movie Manager API.

```
typedef QTRuntimeSpriteDescStruct * QTRuntimeSpriteDescPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

QTRuntimeSpriteDescStruct

Provides a sprite description for the SpriteMediaNewSprite function.

```
QTRuntimeSpriteDescStruct {
     long                                version;
     QTAtomID                            spriteID;
     short                               imageIndex;
     MatrixRecord                        matrix;
     short                               visible;
     short                               layer;
     ModifierTrackGraphicsModeRecord     graphicsMode;
     QTAtomID                            actionHandlingSpriteID;
 };
```

Fields
`version`

Discussion
Set to 0.

`spriteID`

Discussion
The QT atom ID of the sprite atom.

`imageIndex`

Discussion
The index of the sprite image. This value must be between 1 and the number of available images. You can determine how many images are available by calling `SpriteMediaCountImages` (page 313).

`matrix`

Discussion
A `MatrixRecord` structure that defines the sprite's matrix.

`visible`

Discussion
Undocumented

`layer`

Discussion
The sprite's layer number.

`graphicsMode`

Discussion
A `ModifierTrackGraphicsModeRecord` structure that defines the graphics mode setting for the sprite.

`actionHandlingSpriteID`

Discussion
Undocumented

Declared In
`Movies.h`

RegionCode

Represents a type used by the Movie Manager API.

```
typedef SInt16 RegionCode;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacTypes.h`

Style

Represents a type used by the Movie Manager API.

```
typedef unsigned char Style;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacTypes.h`

TEHandle

Represents a type used by the Movie Manager API.

```
typedef TEPtr * TEHandle;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TextEdit.h`

TEPtr

Represents a type used by the Movie Manager API.

```
typedef TERec * TEPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TextEdit.h`

TextDescriptionHandle

Represents a type used by the Movie Manager API.

```
typedef TextDescriptionPtr * TextDescriptionHandle;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

TextDescriptionPtr

Represents a type used by the Movie Manager API.

```
typedef TextDescription * TextDescriptionPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

TimeBaseStatus

Represents a type used by the Movie Manager API.

```
typedef unsigned long TimeBaseStatus;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

Constants

TextMediaFindNextText Values

Constants passed to TextMediaFindNextText.

```
enum {
  findTextEdgeOK              = 1 << 0, /* Okay to find text at specified sample
 time*/
  findTextCaseSensitive       = 1 << 1, /* Case sensitive search*/
  findTextReverseSearch       = 1 << 2, /* Search from sampleTime backwards*/
  findTextWrapAround          = 1 << 3, /* Wrap search when beginning or end of
 movie is hit*/
  findTextUseOffset           = 1 << 4 /* Begin search at the given character
offset into sample rather than edge*/
};
```

Declared In
`Movies.h`

QTTextToNativeText Values

Constants passed to QTTextToNativeText.

```
enum {
  kITextAtomType                = 'itxt',
  kITextStringAtomType          = 'text'
};
```

Declared In
`Movies.h`

ITextRemoveString Values

Constants passed to ITextRemoveString.

```
enum {
  kITextRemoveEverythingBut     = 0 << 1,
  kITextRemoveLeaveSuggestedAlternate = 1 << 1
};
```

Declared In
`Movies.h`

CreateMovieControl Values

Constants passed to CreateMovieControl.

```
enum {
  kMovieControlOptionHideController = (1L << 0),
  kMovieControlOptionLocateTopLeft = (1L << 1),
  kMovieControlOptionEnableEditing = (1L << 2),
  kMovieControlOptionHandleEditingHI = (1L << 3),
  kMovieControlOptionSetKeysEnabled = (1L << 4),
  kMovieControlOptionManuallyIdled = (1L << 5)
};
```

Declared In
`Movies.h`

MovieMediaGetCurrentMovieProperty Values

Constants passed to MovieMediaGetCurrentMovieProperty.

```
enum {
  kMoviePropertyDuration        = 'dura', /* TimeValue **/
  kMoviePropertyTimeScale       = 'tims', /* TimeValue **/
  kMoviePropertyTime            = 'timv', /* TimeValue **/
  kMoviePropertyNaturalBounds   = 'natb', /* Rect **/
  kMoviePropertyMatrix          = 'mtrx', /* Matrix **/
  kMoviePropertyTrackList       = 'tlst' /* long ****/
};
```

Declared In
`Movies.h`

EnterMoviesOnThread Values

Constants passed to EnterMoviesOnThread.

```
enum {
  kQTEnterMoviesFlagDontSetComponentsThreadMode = 1L << 0
};
```

Declared In
`Movies.h`

loopTimeBase

Constants grouped with loopTimeBase.

```
enum {
  loopTimeBase                  = 1,
  palindromeLoopTimeBase        = 2,
  maintainTimeBaseZero          = 4
};
```

Declared In
`Movies.h`

SetMovieDrawingCompleteProc Values

Constants passed to SetMovieDrawingCompleteProc.

```
enum {
  movieDrawingCallWhenChanged   = 0,
  movieDrawingCallAlways        = 1
};
```

Declared In
`Movies.h`

timeBaseAfterStopTime

Constants grouped with timeBaseAfterStopTime.

```
enum {
  timeBaseBeforeStartTime       = 1,
  timeBaseAfterStopTime         = 2,
  timeBaseRateChanging          = 4
};
```

Declared In

`Movies.h`

Constants

QuickTime Constants Reference

Framework:	Frameworks/QuickTime.framework
Declared in	QuickTime.h

Overview

This reference covers the constants common to multiple QuickTime frameworks.

Constants

Atom ID Codes

Identify the four-character type codes of atoms.

```
enum {
  ConnectionSpeedPrefsType      = 'cspd',
  ConnectionSpeedIsValidPrefsType = 'vspd'
};
enum {
  kEffectNameAtom               = 'name', /* name of effect */
  kEffectTypeAtom               = 'type', /* codec sub-type for effect */
  kEffectManufacturerAtom       = 'manu' /* codec manufacturer for effect */
};
enum {
  kGraphicsExportGroup          = 'expo',
  kGraphicsExportFileType       = 'ftyp',
  kGraphicsExportMIMEType       = 'mime',
  kGraphicsExportExtension      = 'ext ',
  kGraphicsExportDescription    = 'desc'
};
enum {
  kInputMapSubInputID           = 'subi'
};
enum {
  kMovieMediaDataReference      = 'mmdr', /* data reference*/
  kMovieMediaDefaultDataReferenceID = 'ddri', /* atom id*/
  kMovieMediaSlaveTime          = 'slti', /* boolean*/
  kMovieMediaSlaveAudio         = 'slau', /* boolean*/
  kMovieMediaSlaveGraphicsMode  = 'slgr', /* boolean*/
  kMovieMediaAutoPlay           = 'play', /* boolean*/
  kMovieMediaLoop              = 'loop', /* UInt8 (0=no loop, 1=loop, 2=palindrome
 loop)*/
  kMovieMediaUseMIMEType        = 'mime', /* string indicating the MIME type to
use for the dataref (usually not required)*/
  kMovieMediaTitle            = 'titl', /* string of the media's title (tooltips)*/
  kMovieMediaAltText            = 'altt', /* string of alternate text if media
isn't loaded*/
  kMovieMediaClipBegin          = 'clpb', /* MovieMediaTimeRecord of start time of
 embedded media*/
  kMovieMediaClipDuration       = 'clpd', /* MovieMediaTimeRecord of duration of
embedded media*/
  kMovieMediaRegionAtom        = 'regi', /* contains subatoms that describe layout*/
  kMovieMediaSlaveTrackDuration = 'sltr', /* Boolean indicating that media handler
 should adjust track and media based on actual embedded movie duration*/
  kMovieMediaEnableFrameStepping = 'enfs', /* boolean. if true stepping on external
 movie steps frames within embedded movie.*/
  kMovieMediaBackgroundColor    = 'bkcl', /* RGBColor.*/
  kMovieMediaPrerollTime        = 'prer' /* SInt32 indicating preroll time*/
};
enum {
  kMovieMediaSpatialAdjustment  = 'fit ', /* OSType from kMovieMediaFit**/
  kMovieMediaRectangleAtom      = 'rect',
  kMovieMediaTop                = 'top ',
  kMovieMediaLeft               = 'left',
  kMovieMediaWidth              = 'wd  ',
  kMovieMediaHeight             = 'ht  '
};
enum {
  kQTEventType                  = 'evnt',
  kAction                       = 'actn',
  kWhichAction                  = 'whic',
  kActionParameter              = 'parm',
  kActionTarget                 = 'targ',
  kActionFlags                  = 'flag',
  kActionParameterMinValue      = 'minv',
  kActionParameterMaxValue      = 'maxv',
  kActionListAtomType           = 'list',
  kExpressionContainerAtomType  = 'expr',
  kConditionalAtomType          = 'test',
  kOperatorAtomType             = 'oper',
  kOperandAtomType              = 'oprn',
  kCommentAtomType              = 'why ',
  kCustomActionHandler          = 'cust',
  kCustomHandlerID              = 'id  ',
  kCustomHandlerDesc            = 'desc',
  kQTEventRecordAtomType        = 'erec'
};
enum {
  kQTParseTextHREFText          = 'text', /* string*/
  kQTParseTextHREFBaseURL       = 'burl', /* string*/
  kQTParseTextHREFClickPoint    = 'clik', /* Point; if present, QTParseTextHREF
will expand URLs to support server-side image maps*/
  kQTParseTextHREFUseAltDelim   = 'altd', /* boolean; if no
kQTParseTextHREFDelimiter, delim is ':'*/
  kQTParseTextHREFDelimiter     = 'delm', /* character*/
  kQTParseTextHREFRecomposeHREF = 'rhrf' /* Boolean; if true, QTParseTextHREF
returns recomposed HREF with URL expanded as appropriate*/
};
enum {
  kQTResolutionSettings         = 'reso',
  kQTTargetDataSize             = 'dasz',
  kQTDontRecompress             = 'dntr',
  kQTInterlaceStyle             = 'ilac',
  kQTColorSyncProfile           = 'iccp',
  kQTThumbnailSettings          = 'thum',
  kQTEnableExif                 = 'exif', /* UInt8 (boolean)*/
  kQTMetaData                   = 'meta'
};
enum {
  kQTSConnectionPrefsType       = 'stcm', /* root atom that all other atoms are
contained in*/
                                     /*    kQTSNotUsedForProxyPrefsType = 'nopr',
     //        comma-delimited list of URLs that are never used for proxies*/
  kQTSConnectionMethodPrefsType = 'mthd', /*      connection method (OSType that
matches one of the following three)*/
  kQTSDirectConnectPrefsType    = 'drct', /*       used if direct connect
(QTSDirectConnectPrefsRecord)*/
                                       /*    kQTSRTSPProxyPrefsType =     'rtsp',
   //   used if RTSP Proxy (QTSProxyPrefsRecord)*/
  kQTSSOCKSPrefsType            = 'sock' /*       used if SOCKS Proxy
(QTSProxyPrefsRecord)*/
};
enum {
  kQTSNullNotification          = 'null', /* NULL */
  kQTSErrorNotification         = 'err ', /* QTSErrorParams*, optional */
  kQTSNewPresDetectedNotification = 'newp', /* QTSNewPresDetectedParams* */
  kQTSPresBeginChangingNotification = 'prcb', /* NULL */
  kQTSPresDoneChangingNotification = 'prcd', /* NULL */
  kQTSPresentationChangedNotification = 'prch', /* NULL */
  kQTSNewStreamNotification     = 'stnw', /* QTSNewStreamParams* */
  kQTSStreamBeginChangingNotification = 'stcb', /* QTSStream */
  kQTSStreamDoneChangingNotification = 'stcd', /* QTSStream */
  kQTSStreamChangedNotification = 'stch', /* QTSStreamChangedParams* */
  kQTSStreamGoneNotification    = 'stgn', /* QTSStreamGoneParams* */
  kQTSPreviewAckNotification    = 'pvak', /* QTSStream */
  kQTSPrerollAckNotification    = 'pack', /* QTSStream */
  kQTSStartAckNotification      = 'sack', /* QTSStream */
  kQTSStopAckNotification       = 'xack', /* QTSStream */
  kQTSStatusNotification        = 'stat', /* QTSStatusParams* */
  kQTSURLNotification           = 'url ', /* QTSURLParams* */
  kQTSDurationNotification      = 'dura', /* QTSDurationAtom* */
  kQTSNewPresentationNotification = 'nprs', /* QTSPresentation */
  kQTSPresentationGoneNotification = 'xprs', /* QTSPresentation */
  kQTSPresentationDoneNotification = 'pdon', /* NULL */
  kQTSBandwidthAlertNotification = 'bwal', /* QTSBandwidthAlertParams* */
  kQTSAnnotationsChangedNotification = 'meta' /* NULL */
};
enum {
  kQTSStatisticsInfo            = 'stat', /* QTSStatisticsParams* */
  kQTSMinStatusDimensionsInfo   = 'mstd', /* QTSDimensionParams* */
  kQTSNormalStatusDimensionsInfo = 'nstd', /* QTSDimensionParams* */
  kQTSTotalDataRateInfo         = 'drtt', /* UInt32*, add to what's there */
  kQTSTotalDataRateInInfo       = 'drti', /* UInt32*, add to what's there */
  kQTSTotalDataRateOutInfo      = 'drto', /* UInt32*, add to what's there */
  kQTSLostPercentInfo           = 'lpct', /* QTSLostPercentParams*, add to what's
 there */
  kQTSNumViewersInfo            = 'nviw', /* UInt32* */
  kQTSMediaTypeInfo             = 'mtyp', /* OSType* */
  kQTSNameInfo                  = 'name', /* QTSNameParams* */
  kQTSCanHandleSendDataType     = 'chsd', /* QTSCanHandleSendDataTypeParams* */
  kQTSAnnotationsInfo           = 'meta', /* QTAtomContainer */
  kQTSRemainingBufferTimeInfo   = 'btms', /* UInt32* remaining buffer time before
 playback, in microseconds */
  kQTSInfo_SettingsText         = 'sttx', /* QTSSettingsTextParams* */
  kQTSInfo_AverageFrameRate     = 'fps ' /* UnsignedFixed* */
};
enum {
  kQTSStreamMediaType           = 'strm'
};
enum {
  kQTSTargetBufferDurationInfo  = 'bufr', /* Fixed* in seconds; expected, not actual
 */
  kQTSDurationInfo              = 'dura', /* QTSDurationAtom* */
  kQTSSoundLevelMeteringEnabledInfo = 'mtrn', /* Boolean* */
  kQTSSoundLevelMeterInfo       = 'levm', /* LevelMeterInfoPtr */
  kQTSSourceTrackIDInfo         = 'otid', /* UInt32* */
  kQTSSourceLayerInfo           = 'olyr', /* UInt16* */
  kQTSSourceLanguageInfo        = 'olng', /* UInt16* */
  kQTSSourceTrackFlagsInfo      = 'otfl', /* SInt32* */
  kQTSSourceDimensionsInfo      = 'odim', /* QTSDimensionParams* */
  kQTSSourceVolumesInfo         = 'ovol', /* QTSVolumesParams* */
  kQTSSourceMatrixInfo          = 'omat', /* MatrixRecord* */
  kQTSSourceClipRectInfo        = 'oclp', /* Rect* */
  kQTSSourceGraphicsModeInfo    = 'ogrm', /* QTSGraphicsModeParams* */
  kQTSSourceScaleInfo           = 'oscl', /* Point* */
  kQTSSourceBoundingRectInfo    = 'orct', /* Rect* */
  kQTSSourceUserDataInfo        = 'oudt', /* UserData */
  kQTSSourceInputMapInfo        = 'oimp', /* QTAtomContainer */
  kQTSInfo_DataProc               = 'datp', /* QTSDataProcParams* */
  kQTSInfo_SendDataExtras         = 'dext', /* QTSSendDataExtrasParams* */
  kQTSInfo_HintTrackID            = 'htid', /* long* */
  kQTSInfo_URL                    = 'url ', /* Handle*, cstring in handle */
  kQTSInfo_Authentication         = 'auup', /* QTSAuthenticationParams */
  kQTSInfo_MediaPacketizer        = 'rmpk' /* ComponentInstance */
};
enum {
  kQTVRNodeHeaderAtomType         = 'ndhd',
  kQTVRHotSpotParentAtomType      = 'hspa',
  kQTVRHotSpotAtomType            = 'hots',
  kQTVRHotSpotInfoAtomType        = 'hsin',
  kQTVRLinkInfoAtomType           = 'link'
};
enum {
  kQTVRObjectInfoAtomID           = 1,
  kQTVRObjectImageTrackRefAtomID = 1,   /* New with 2.1, it adds a track reference
 to select between multiple image tracks*/
  kQTVRObjectHotSpotTrackRefAtomID = 1  /* New with 2.1, it adds a track reference
 to select between multiple hotspot tracks*/
};
enum {
  kQTVRStringAtomType             = 'vrsg',
  kQTVRStringEncodingAtomType     = 'vrse', /* New with 2.1*/
  kQTVRPanoSampleDataAtomType     = 'pdat',
  kQTVRObjectInfoAtomType         = 'obji',
  kQTVRImageTrackRefAtomType      = 'imtr', /* Parent is kQTVRObjectInfoAtomType.
Required if track ref is not 1 as required by 2.0 format.*/
  kQTVRHotSpotTrackRefAtomType    = 'hstr', /* Parent is kQTVRObjectInfoAtomType.
Required if track ref is not 1 as required by 2.0 format.*/
  kQTVRAngleRangeAtomType         = 'arng',
  kQTVRTrackRefArrayAtomType      = 'tref',
  kQTVRPanConstraintAtomType      = 'pcon',
  kQTVRTiltConstraintAtomType     = 'tcon',
  kQTVRFOVConstraintAtomType      = 'fcon',
  kQTVRCubicViewAtomType          = 'cuvw', /* New with 5.0*/
  kQTVRCubicFaceDataAtomType      = 'cufa' /* New with 5.0*/
};
enum {
  kQTVRWorldHeaderAtomType        = 'vrsc',
  kQTVRImagingParentAtomType      = 'imgp',
  kQTVRPanoImagingAtomType        = 'impn',
  kQTVRObjectImagingAtomType      = 'imob',
  kQTVRNodeParentAtomType         = 'vrnp',
  kQTVRNodeIDAtomType             = 'vrni',
  kQTVRNodeLocationAtomType       = 'nloc',
  kQTVRCursorParentAtomType       = 'vrcp', /* New with 2.1*/
  kQTVRCursorAtomType             = 'CURS', /* New with 2.1*/
  kQTVRColorCursorAtomType        = 'crsr' /* New with 2.1*/
};
enum {
  kSpriteAtomType                 = 'sprt',
  kSpriteImagesContainerAtomType = 'imct',
  kSpriteImageAtomType            = 'imag',
  kSpriteImageDataAtomType        = 'imda',
  kSpriteImageDataRefAtomType     = 'imre',
  kSpriteImageDataRefTypeAtomType = 'imrt',
  kSpriteImageGroupIDAtomType     = 'imgr',
  kSpriteImageRegistrationAtomType = 'imrg',
  kSpriteImageDefaultImageIndexAtomType = 'defi',
  kSpriteSharedDataAtomType     = 'dflt',
  kSpriteNameAtomType           = 'name',
  kSpriteImageNameAtomType      = 'name',
  kSpriteUsesImageIDsAtomType   = 'uses', /* leaf data is an array of QTAtomID's,
 one per image used*/
  kSpriteBehaviorsAtomType      = 'beha',
  kSpriteImageBehaviorAtomType  = 'imag',
  kSpriteCursorBehaviorAtomType = 'crsr',
  kSpriteStatusStringsBehaviorAtomType = 'sstr',
  kSpriteVariablesContainerAtomType = 'vars',
  kSpriteStringVariableAtomType = 'strv',
  kSpriteFloatingPointVariableAtomType = 'flov'
};
enum {
  kTargetMovie                  = 'moov', /* no data */
  kTargetMovieName              = 'mona', /* (PString movieName) */
  kTargetMovieID                = 'moid', /* (long movieID) */
  kTargetRootMovie              = 'moro', /* no data */
  kTargetParentMovie            = 'mopa', /* no data */
  kTargetChildMovieTrackName    = 'motn', /* (PString childMovieTrackName) */
  kTargetChildMovieTrackID      = 'moti', /* (long childMovieTrackID) */
  kTargetChildMovieTrackIndex   = 'motx', /* (long childMovieTrackIndex) */
  kTargetChildMovieMovieName    = 'momn', /* (PString childMovieName) */
  kTargetChildMovieMovieID      = 'momi', /* (long childMovieID) */
  kTargetTrackName              = 'trna', /* (PString trackName) */
  kTargetTrackID                = 'trid', /* (long trackID) */
  kTargetTrackType              = 'trty', /* (OSType trackType) */
  kTargetTrackIndex             = 'trin', /* (long trackIndex) */
  kTargetSpriteName             = 'spna', /* (PString spriteName) */
  kTargetSpriteID               = 'spid', /* (QTAtomID spriteID) */
  kTargetSpriteIndex            = 'spin', /* (short spriteIndex) */
  kTargetQD3DNamedObjectName    = 'nana', /* (CString objectName) */
  kTargetCurrentQTEventParams   = 'evpa' /* no data */
};
enum {
  kTrackModifierInput           = 0x696E, /* is really 'in'*/
  kTrackModifierType            = 0x7479, /* is really 'ty'*/
  kTrackModifierReference       = 'ssrc',
  kTrackModifierObjectID        = 'obid',
  kTrackModifierInputName       = 'name'
};
enum {
  kTrackPropertyMediaType       = 'mtyp', /* OSType*/
  kTrackPropertyInstantiation   = 'inst' /* MovieMediaInstantiationInfoRecord*/
};
enum {
  kTrackReferenceChapterList    = 'chap',
  kTrackReferenceTimeCode       = 'tmcd',
  kTrackReferenceModifier       = 'ssrc'
};
enum {
  kTweenEntry                   = 'twen',
  kTweenData                    = 'data',
  kTweenType                    = 'twnt',
  kTweenStartOffset             = 'twst',
  kTweenDuration                = 'twdu',
  kTweenFlags                  = 'flag',
  kTweenOutputMin              = 'omin',
  kTweenOutputMax              = 'omax',
  kTweenSequenceElement        = 'seqe',
  kTween3dInitialCondition     = 'icnd',
  kTweenInterpolationID        = 'intr',
  kTweenRegionData             = 'qdrg',
  kTweenPictureData            = 'PICT',
  kListElementType             = 'type',
  kListElementDataType         = 'daty',
  kNameAtom                    = 'name',
  kInitialRotationAtom         = 'inro',
  kNonLinearTweenHeader        = 'nlth'
};
enum {
  MovieAID                     = 'moov',
  MovieHeaderAID               = 'mvhd',
  ClipAID                      = 'clip',
  RgnClipAID                   = 'crgn',
  MatteAID                     = 'matt',
  MatteCompAID                 = 'kmat',
  TrackAID                     = 'trak',
  UserDataAID                  = 'udta',
  TrackHeaderAID               = 'tkhd',
  EditsAID                     = 'edts',
  EditListAID                  = 'elst',
  MediaAID                     = 'mdia',
  MediaHeaderAID               = 'mdhd',
  MediaInfoAID                 = 'minf',
  VideoMediaInfoHeaderAID      = 'vmhd',
  SoundMediaInfoHeaderAID      = 'smhd',
  GenericMediaInfoHeaderAID    = 'gmhd',
  GenericMediaInfoAID          = 'gmin',
  DataInfoAID                  = 'dinf',
  DataRefAID                   = 'dref',
  SampleTableAID               = 'stbl',
  STSampleDescAID              = 'stsd',
  STTimeToSampAID              = 'stts',
  STSyncSampleAID              = 'stss',
  STSampleToChunkAID           = 'stsc',
  STShadowSyncAID              = 'stsh',
  HandlerAID                   = 'hdlr',
  STSampleSizeAID              = 'stsz',
  STChunkOffsetAID             = 'stco',
  STChunkOffset64AID           = 'co64',
  STSampleIDAID                = 'stid',
  STCompositionOffsetAID       = 'ctts',
  STSampleDependencyAID        = 'sdtp',
  STCompositionShiftLeastGreatestAID = 'cslg',
  STPartialSyncSampleAID       = 'stps',
  DataRefContainerAID          = 'drfc',
  TrackReferenceAID            = 'tref',
  ColorTableAID                = 'ctab',
  LoadSettingsAID              = 'load',
  PropertyAtomAID              = 'code',
  InputMapAID                  = 'imap',
  MovieBufferHintsAID          = 'mbfh',
  MovieDataRefAliasAID         = 'mdra',
  SoundLocalizationAID          = 'sloc',
  CompressedMovieAID            = 'cmov',
  CompressedMovieDataAID        = 'cmvd',
  DataCompressionAtomAID        = 'dcom',
  ReferenceMovieRecordAID       = 'rmra',
  ReferenceMovieDescriptorAID   = 'rmda',
  ReferenceMovieDataRefAID      = 'rdrf',
  ReferenceMovieVersionCheckAID = 'rmvc',
  ReferenceMovieDataRateAID     = 'rmdr',
  ReferenceMovieComponentCheckAID = 'rmcd',
  ReferenceMovieQualityAID      = 'rmqu',
  ReferenceMovieLanguageAID     = 'rmla',
  ReferenceMovieCPURatingAID    = 'rmcs',
  ReferenceMovieAlternateGroupAID = 'rmag',
  ReferenceMovieNetworkStatusAID = 'rnet',
  CloneMediaAID                 = 'clon',
  FileTypeAID                   = 'ftyp',
  SecureContentInfoAID          = 'sinf',
  SecureContentSchemeTypeAID    = 'schm',
  SecureContentSchemeInfoAID    = 'schi'
};
enum {
  MovieResourceAtomType         = 'moov',
  MovieDataAtomType             = 'mdat',
  FreeAtomType                  = 'free',
  SkipAtomType                  = 'skip',
  WideAtomPlaceholderType       = 'wide'
};
enum {
  quickTimeImageFileImageDescriptionAtom = 'idsc',
  quickTimeImageFileImageDataAtom = 'idat',
  quickTimeImageFileMetaDataAtom = 'meta',
  quickTimeImageFileColorSyncProfileAtom = 'iicc'
};
```

Constants

`kMovieMediaDefaultDataReferenceID`

Atom id.

Available in Mac OS X v10.0 and later.

Declared in `Movies.h`.

`kMovieMediaSlaveTime`

Boolean.

Available in Mac OS X v10.0 and later.

Declared in `Movies.h`.

`kMovieMediaSlaveGraphicsMode`

Boolean.

Available in Mac OS X v10.0 and later.

Declared in `Movies.h`.

`kMovieMediaBackgroundColor`

RGBColor..

Available in Mac OS X v10.0 and later.

Declared in `Movies.h`.

`kMovieMediaPrerollTime`
: SInt32 indicating preroll time.

  Available in Mac OS X v10.0 and later.

  Declared in `Movies.h`.

`kQTParseTextHREFText`
: String.

  Available in Mac OS X v10.0 and later.

  Declared in `Movies.h`.

`kQTEnableExif`
: UInt8 (Boolean).

  Available in Mac OS X v10.1 and later.

  Declared in `ImageCompression.h`.

`kTargetChildMovieTrackIndex`
: (long childMovieTrackIndex).

  Available in Mac OS X v10.0 and later.

  Declared in `Movies.h`.

`kTargetChildMovieMovieName`
: (PString childMovieName).

  Available in Mac OS X v10.0 and later.

  Declared in `Movies.h`.

`kTargetTrackType`
: (OSType trackType).

  Available in Mac OS X v10.0 and later.

  Declared in `Movies.h`.

`kTargetTrackIndex`
: (long trackIndex).

  Available in Mac OS X v10.0 and later.

  Declared in `Movies.h`.

`kTargetSpriteName`
: (PString spriteName).

  Available in Mac OS X v10.0 and later.

  Declared in `Movies.h`.

`kTargetSpriteID`
: (QTAtomID spriteID).

  Available in Mac OS X v10.0 and later.

  Declared in `Movies.h`.

`kTargetQD3DNamedObjectName`
: (CString objectName).

  Available in Mac OS X v10.0 and later.

  Declared in `Movies.h`.

`kTargetCurrentQTEventParams`
: No data.

  Available in Mac OS X v10.0 and later.

  Declared in `Movies.h`.

FCompressImage Values

Constants passed to FCompressImage.

```
enum {
  codecFlagUseImageBuffer       = (1L << 0), /* decompress*/
  codecFlagUseScreenBuffer      = (1L << 1), /* decompress*/
  codecFlagUpdatePrevious       = (1L << 2), /* compress*/
  codecFlagNoScreenUpdate       = (1L << 3), /* decompress*/
  codecFlagWasCompressed        = (1L << 4), /* compress*/
  codecFlagDontOffscreen        = (1L << 5), /* decompress*/
  codecFlagUpdatePreviousComp   = (1L << 6), /* compress*/
  codecFlagForceKeyFrame        = (1L << 7), /* compress*/
  codecFlagOnlyScreenUpdate     = (1L << 8), /* decompress*/
  codecFlagLiveGrab             = (1L << 9), /* compress*/
  codecFlagDiffFrame            = (1L << 9), /* decompress*/
  codecFlagDontUseNewImageBuffer = (1L << 10), /* decompress*/
  codecFlagInterlaceUpdate      = (1L << 11), /* decompress*/
  codecFlagCatchUpDiff          = (1L << 12), /* decompress*/
  codecFlagSupportDisable       = (1L << 13), /* decompress*/
  codecFlagReenable             = (1L << 14) /* decompress*/
};
```

Constants

`codecFlagUpdatePrevious`
: Controls whether your compressor updates the previous image during compression. This flag is only used with sequences that are being temporally compressed. If this flag is set to 1, your compressor should copy the current frame into the previous frame buffer at the end of the frame-compression sequence. Use the source image.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCompression.h`.

`codecFlagWasCompressed`
: Indicates to your compressor that the image to be compressed has been compressed before. This information may be useful to compressors that can compensate for the image degradation that may otherwise result from repeated compression and decompression of the same image. This flag is set to 1 to indicate that the image was previously compressed. This flag is set to 0 if the image was not previously compressed.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCompression.h`.

`codecFlagUpdatePreviousComp`
: Controls whether your compressor updates the previous image buffer with the compressed image. This flag is only used with temporal compression. If this flag is set to 1, your compressor should update the previous frame buffer at the end of the frame-compression sequence, allowing your compressor to perform frame differencing against the compression results. Use the image that results from the compression operation. If this flag is set to 0, your compressor should not modify the previous frame buffer during compression.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCompression.h`.

`codecFlagLiveGrab`
: Indicates whether the current sequence results from grabbing live video. When working with live video, your compressor should operate as quickly as possible and disable any additional processing, such as compensation for previously compressed data. This flag is set to 1 when you are compressing from a live video source.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCompression.h`.

`codecFlagDiffFrame`
: Decompress.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCompression.h`.

`codecFlagSupportDisable`
: Decompress.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCompression.h`.

Codec Flags

Constants that represent codec flags.

```
enum {
  codecCompletionSource         = (1 << 0), /* asynchronous codec is done with
source data */
  codecCompletionDest           = (1 << 1), /* asynchronous codec is done with
destination data */
  codecCompletionDontUnshield   = (1 << 2), /* on dest complete don't unshield
cursor */
  codecCompletionWentOffscreen  = (1 << 3), /* codec used offscreen buffer */
  codecCompletionUnlockBits     = (1 << 4), /* on dest complete, call
ICMSequenceUnlockBits */
  codecCompletionForceChainFlush = (1 << 5), /* ICM needs to flush the whole chain
 */
  codecCompletionDropped        = (1 << 6), /* codec decided to drop this frame */
  codecCompletionDecoded        = (1 << 10), /* codec has decoded this frame; if
it is cancelled and rescheduled, set icmFrameAlreadyDecoded in
ICMFrameTimeRecord.flags */
  codecCompletionNotDisplayable = (1 << 11), /* the frame may still be scheduled
for decode, but will not be able to be displayed because the buffer containing it
 will need to be recycled to display earlier frames. */
  codecCompletionNotDrawn       = (1 << 12) /* set in conjunction with
codecCompletionDest to indicate that the frame was not drawn */
};
enum {
  codecFlagOutUpdateOnNextIdle  = (1L << 9),
  codecFlagOutUpdateOnDataSourceChange = (1L << 10),
  codecFlagSequenceSensitive    = (1L << 11),
  codecFlagOutUpdateOnTimeChange = (1L << 12),
  codecFlagImageBufferNotSourceImage = (1L << 13),
  codecFlagUsedNewImageBuffer   = (1L << 14),
  codecFlagUsedImageBuffer      = (1L << 15)
};
enum {
  codecInfoDoes1                = (1L << 0), /* codec can work with 1-bit pixels
*/
  codecInfoDoes2                = (1L << 1), /* codec can work with 2-bit pixels
*/
  codecInfoDoes4                = (1L << 2), /* codec can work with 4-bit pixels
*/
  codecInfoDoes8                = (1L << 3), /* codec can work with 8-bit pixels
*/
  codecInfoDoes16               = (1L << 4), /* codec can work with 16-bit pixels
 */
  codecInfoDoes32               = (1L << 5), /* codec can work with 32-bit pixels
 */
  codecInfoDoesDither           = (1L << 6), /* codec can do ditherMode */
  codecInfoDoesStretch          = (1L << 7), /* codec can stretch to arbitrary
sizes */
  codecInfoDoesShrink           = (1L << 8), /* codec can shrink to arbitrary sizes
 */
  codecInfoDoesMask             = (1L << 9), /* codec can mask to clipping regions
 */
  codecInfoDoesTemporal         = (1L << 10), /* codec can handle temporal redundancy
 */
  codecInfoDoesDouble           = (1L << 11), /* codec can stretch to double size
 exactly */
  codecInfoDoesQuad             = (1L << 12), /* codec can stretch to quadruple
size exactly */
  codecInfoDoesHalf             = (1L << 13), /* codec can shrink to half size */
  codecInfoDoesQuarter          = (1L << 14), /* codec can shrink to quarter size
 */
  codecInfoDoesRotate           = (1L << 15), /* codec can rotate on decompress */
  codecInfoDoesHorizFlip        = (1L << 16), /* codec can flip horizontally on
decompress */
  codecInfoDoesVertFlip         = (1L << 17), /* codec can flip vertically on
decompress */
  codecInfoHasEffectParameterList = (1L << 18), /* codec implements get effects
parameter list call, once was codecInfoDoesSkew */
  codecInfoDoesBlend            = (1L << 19), /* codec can blend on decompress */
  codecInfoDoesReorder          = (1L << 19), /* codec can rearrange frames during
 compression */
  codecInfoDoesWarp             = (1L << 20), /* codec can warp arbitrarily on
decompress */
  codecInfoDoesMultiPass        = (1L << 20), /* codec can perform multi-pass
compression */
  codecInfoDoesRecompress       = (1L << 21), /* codec can recompress image without
 accumulating errors */
  codecInfoDoesSpool            = (1L << 22), /* codec can spool image data */
  codecInfoDoesRateConstrain    = (1L << 23) /* codec can data rate constrain */
};
enum {
  codecLockBitsShieldCursor     = (1 << 0) /* shield cursor */
};
```

Constants

`codecCompletionSource`
: The Image Compression Manager is done with the source buffer. The Image Compression Manager sets this flag to 1 when it is done with the processing associated with the source buffer. For compression operations, the source is the uncompressed pixel map you are compressing. For decompression operations, the source is the decompressed data you are decompressing.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCompression.h`.

`codecCompletionDest`
: The Image Compression Manager is done with the destination buffer. The Image Compression Manager sets this flag to 1 when it is done with the processing associated with the destination buffer.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCompression.h`.

`codecCompletionWentOffscreen`
: Codec used offscreen buffer.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCompression.h`.

`codecCompletionUnlockBits`
: On dest complete, call ICMSequenceUnlockBits.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCompression.h`.

`codecCompletionForceChainFlush`
: ICM needs to flush the whole chain.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCompression.h`.

`codecCompletionDropped`
: Codec decided to drop this frame.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCompression.h`.

`codecCompletionDecoded`
: Codec has decoded this frame; if it is cancelled and rescheduled, set icmFrameAlreadyDecoded in ICMFrameTimeRecord.flags.

  Available in Mac OS X v10.3 and later.

  Declared in `ImageCompression.h`.

`codecCompletionNotDisplayable`
: The frame may still be scheduled for decode, but will not be able to be displayed because the buffer containing it will need to be recycled to display earlier frames..

  Available in Mac OS X v10.3 and later.

  Declared in `ImageCompression.h`.

`codecCompletionNotDrawn`
: Set in conjunction with codecCompletionDest to indicate that the frame was not drawn.

  Available in Mac OS X v10.3 and later.

  Declared in `ImageCompression.h`.

`codecFlagUsedImageBuffer`
: Indicates to your application that the decompressor used the offscreen image buffer for this frame. If this flag is set to 1, the decompressor used the image buffer. If this flag is set to 0, the decompressor did not use the image buffer.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCompression.h`.

`codecInfoDoes1`
: Codec can work with 1-bit pixels.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCompression.h`.

`codecInfoDoes2`
: Codec can work with 2-bit pixels.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCompression.h`.

`codecInfoDoes4`
: Codec can work with 4-bit pixels.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCompression.h`.

`codecInfoDoes8`
: Codec can work with 8-bit pixels.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCompression.h`.

`codecInfoDoes16`
Codec can work with 16-bit pixels.
Available in Mac OS X v10.0 and later.
Declared in `ImageCompression.h`.

`codecInfoDoes32`
Codec can work with 32-bit pixels.
Available in Mac OS X v10.0 and later.
Declared in `ImageCompression.h`.

`codecInfoDoesDither`
Codec can dither images.
Available in Mac OS X v10.0 and later.
Declared in `ImageCompression.h`.

`codecInfoDoesStretch`
Codec can stretch images to arbitrary sizes.
Available in Mac OS X v10.0 and later.
Declared in `ImageCompression.h`.

`codecInfoDoesShrink`
Codec can shrink images to arbitrary sizes.
Available in Mac OS X v10.0 and later.
Declared in `ImageCompression.h`.

`codecInfoDoesMask`
Codec can mask images to clipping regions.
Available in Mac OS X v10.0 and later.
Declared in `ImageCompression.h`.

`codecInfoDoesTemporal`
Codec can handle temporal redundancy.
Available in Mac OS X v10.0 and later.
Declared in `ImageCompression.h`.

`codecInfoDoesDouble`
Codec can stretch images to exactly double size.
Available in Mac OS X v10.0 and later.
Declared in `ImageCompression.h`.

`codecInfoDoesQuad`
Codec can stretch images to exactly quadruple size.
Available in Mac OS X v10.0 and later.
Declared in `ImageCompression.h`.

`codecInfoDoesHalf`
Codec can shrink images to exactly half size.
Available in Mac OS X v10.0 and later.
Declared in `ImageCompression.h`.

`codecInfoDoesQuarter`
: Codec can shrink images to exactly quarter size.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCompression.h`.

`codecInfoDoesRotate`
: Codec can rotate images during decompression.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCompression.h`.

`codecInfoDoesHorizFlip`
: Codec can flip images horizontally during decompression.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCompression.h`.

`codecInfoDoesVertFlip`
: Codec can flip images vertically during decompression.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCompression.h`.

`codecInfoHasEffectParameterList`
: Codec implements QTGetEffectsList.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCompression.h`.

`codecInfoDoesBlend`
: Codec can blend image during decompression.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCompression.h`.

`codecInfoDoesReorder`
: Codec can rearrange frames during compression.

  Available in Mac OS X v10.3 and later.

  Declared in `ImageCompression.h`.

`codecInfoDoesWarp`
: Codec can warp image arbitrarily during decompression.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCompression.h`.

`codecInfoDoesMultiPass`
: Codec can perform multi-pass compression.

  Available in Mac OS X v10.3 and later.

  Declared in `ImageCompression.h`.

`codecInfoDoesRecompress`
: Codec can recompress image without accumulating errors.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCompression.h`.

`codecInfoDoesSpool`
Codec can spool image data.

Available in Mac OS X v10.0 and later.

Declared in `ImageCompression.h`.

VDSetCompression Values

Constants passed to VDSetCompression.

```
enum {
  codecLosslessQuality          = 0x00000400,
  codecMaxQuality               = 0x000003FF,
  codecMinQuality               = 0x00000000,
  codecLowQuality               = 0x00000100,
  codecNormalQuality            = 0x00000200,
  codecHighQuality              = 0x00000300
};
```

Constants

`codecLosslessQuality`
Lossless compression or decompression. This special value is valid only for components that can support lossless compression or decompression.

Available in Mac OS X v10.0 and later.

Declared in `ImageCompression.h`.

`codecMaxQuality`
The maximum standard value.

Available in Mac OS X v10.0 and later.

Declared in `ImageCompression.h`.

`codecMinQuality`
The minimum valid value.

Available in Mac OS X v10.0 and later.

Declared in `ImageCompression.h`.

`codecLowQuality`
Low-quality image reproduction. This value should correspond to the lowest image quality that still results in acceptable display characteristics.

Available in Mac OS X v10.0 and later.

Declared in `ImageCompression.h`.

`codecNormalQuality`
Image reproduction of normal quality.

Available in Mac OS X v10.0 and later.

Declared in `ImageCompression.h`.

CodecInfo Values

Constants passed to CodecInfo.

```
enum {
  codecInfoDepth1               = (1L << 0), /* compressed data at 1 bpp depth
available */
  codecInfoDepth2               = (1L << 1), /* compressed data at 2 bpp depth
available */
  codecInfoDepth4               = (1L << 2), /* compressed data at 4 bpp depth
available */
  codecInfoDepth8               = (1L << 3), /* compressed data at 8 bpp depth
available */
  codecInfoDepth16              = (1L << 4), /* compressed data at 16 bpp depth
available */
  codecInfoDepth32              = (1L << 5), /* compressed data at 32 bpp depth
available */
  codecInfoDepth24              = (1L << 6), /* compressed data at 24 bpp depth
available */
  codecInfoDepth33             = (1L << 7), /* compressed data at 1 bpp monochrome
 depth  available */
  codecInfoDepth34              = (1L << 8), /* compressed data at 2 bpp grayscale
 depth available */
  codecInfoDepth36              = (1L << 9), /* compressed data at 4 bpp grayscale
 depth available */
  codecInfoDepth40             = (1L << 10), /* compressed data at 8 bpp grayscale
 depth available */
  codecInfoStoresClut           = (1L << 11), /* compressed data can have custom
cluts */
  codecInfoDoesLossless         = (1L << 12), /* compressed data can be stored in
 lossless format */
  codecInfoSequenceSensitive    = (1L << 13) /* compressed data is sensitive to
out of sequence decoding */
};
```

Constants

`codecInfoDepth1`

Compressed data available at 1 bit-per-pixel depth.

Available in Mac OS X v10.0 and later.

Declared in `ImageCompression.h`.

`codecInfoDepth2`

Compressed data available at 2 bit-per-pixel depth.

Available in Mac OS X v10.0 and later.

Declared in `ImageCompression.h`.

`codecInfoDepth4`

Compressed data available at 4 bit-per-pixel depth.

Available in Mac OS X v10.0 and later.

Declared in `ImageCompression.h`.

`codecInfoDepth8`

Compressed data available at 8 bit-per-pixel depth.

Available in Mac OS X v10.0 and later.

Declared in `ImageCompression.h`.

`codecInfoDepth16`
: Compressed data available at 16 bit-per-pixel depth.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCompression.h`.

`codecInfoDepth32`
: Compressed data available at 32 bit-per-pixel depth.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCompression.h`.

`codecInfoDepth24`
: Compressed data available at 24 bit-per-pixel depth.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCompression.h`.

`codecInfoDepth33`
: Compressed data available at 1 bit-per-pixel monochrome depth.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCompression.h`.

`codecInfoDepth34`
: Compressed data available at 2 bit-per-pixel grayscale depth.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCompression.h`.

`codecInfoDepth36`
: Compressed data available at 4 bit-per-pixel grayscale depth.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCompression.h`.

`codecInfoDepth40`
: Compressed data available at 8 bit-per-pixel grayscale depth.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCompression.h`.

`codecInfoStoresClut`
: Compressed data can have custom color lookup tables.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCompression.h`.

`codecInfoDoesLossless`
: Compressed data can be stored in lossless format.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCompression.h`.

CreateMovieFile Values

Constants passed to CreateMovieFile.

```
enum {
  createMovieFileDeleteCurFile  = 1L << 31,
  createMovieFileDontCreateMovie = 1L << 30,
  createMovieFileDontOpenFile   = 1L << 29,
  createMovieFileDontCreateResFile = 1L << 28
};
```

Constants

`createMovieFileDontOpenFile`

Controls whether the function opens the new movie file. If you set this flag to 1, the Movie Toolbox does not open the new movie file. In this case, the function ignores the outDataHandler parameter. If you set this flag to 0, the Movie Toolbox opens the new movie file and returns its reference number into the field referenced by outDataHandler.

Available in Mac OS X v10.0 and later.

Declared in `Movies.h`.

FlattenMovieData Values

Constants passed to FlattenMovieData.

```
enum {
  flattenAddMovieToDataFork     = 1L << 0,
  flattenActiveTracksOnly       = 1L << 2,
  flattenDontInterleaveFlatten  = 1L << 3,
  flattenFSSpecPtrIsDataRefRecordPtr = 1L << 4,
  flattenCompressMovieResource  = 1L << 5,
  flattenForceMovieResourceBeforeMovieData = 1L << 6
};
```

ICM Preferences and Flags

Constants that represent the flags and preferences for ICM sessions.

```
enum {
  icmFrameTimeHasVirtualStartTimeAndDuration = 1 << 0,
  icmFrameAlreadyDecoded        = 1 << 1,
  icmFrameTimeIsNonScheduledDisplayTime = 1 << 2,
  icmFrameTimeHasDecodeTime     = 1 << 3,
  icmFrameTimeDecodeImmediately = 1 << 4,
  icmFrameTimeDoNotDisplay      = 1 << 5
};
enum {
  kICMGetChainUltimateParent    = 0,
  kICMGetChainParent            = 1,
  kICMGetChainChild             = 2,
  kICMGetChainUltimateChild     = 3
};
enum {
  kICMImageBufferNoPreference   = 0,
  kICMImageBufferPreferMainMemory = 1,
  kICMImageBufferPreferVideoMemory = 2
};
enum {
  kICMNoDeinterlacing           = 0,
  kICMDeinterlaceFields         = 1
};
enum {
  kICMPixelFormatIsPlanarMask   = 0x0F, /* these bits in formatFlags indicate how
 many planes there are; they're 0 if chunky*/
  kICMPixelFormatIsIndexed      = (1L << 4),
  kICMPixelFormatIsSupportedByQD = (1L << 5),
  kICMPixelFormatIsMonochrome   = (1L << 6),
  kICMPixelFormatHasAlphaChannel = (1L << 7)
};
enum {
  kICMSequenceTaskWeight        = 'twei', /* data is pointer to UInt32*/
  kICMSequenceTaskName          = 'tnam', /* data is pointer to OSType*/
  kICMSequenceUserPreferredCodecs = 'punt' /* data is pointer to
CodecComponentHandle*/
};
enum {
  kICMTempThenAppMemory         = 1L << 12,
  kICMAppThenTempMemory         = 1L << 13
};
```

Constants

`icmFrameTimeHasVirtualStartTimeAndDuration`

Indicates that virtualStartTime and virtualDuration are valid.

Available in Mac OS X v10.0 and later.

Declared in `ImageCompression.h`.

`icmFrameTimeHasDecodeTime`

Indicates that decodeTime is valid.

Available in Mac OS X v10.3 and later.

Declared in `ImageCompression.h`.

`kICMPixelFormatIsPlanarMask`

If this flag is 1, the pixel format is a planar mask and bitsPerPixel[] represents the bits for each pixel component. If this flag is 0, the pixel format is chunky (not planar) and bitsPerPixel[0] represents the bits per pixel. Chunky pixel formats pack the different components together. For example, 3 pixels of 32-bit ARGB is represented in memory as ARGBARGBARGB. Planar formats pack the different components separately. If the pixel format is planar, then (formatFlags & kICMPixelFormatIsPlanarMask) is equal to the number of components.

Available in Mac OS X v10.0 and later.

Declared in `ImageCompression.h`.

`kICMPixelFormatIsIndexed`

If the pixel format is indexed (which, by definition, means that there are no individual components) then this flag is 1. Generally, color modes of 8 bit per pixel or less are indexed.

Available in Mac OS X v10.0 and later.

Declared in `ImageCompression.h`.

`kICMPixelFormatIsSupportedByQD`

If this flag is 1, you can call QuickDraw on PixMap structures that store this kind of pixel data. With Macintosh, the classic QD pixel formats will have this set, but not any of the YUV pixel formats. With Windows, more formats will have this set, because the Windows implementation of QuickDraw needs to support more pixel formats.

Available in Mac OS X v10.0 and later.

Declared in `ImageCompression.h`.

`kICMSequenceUserPreferredCodecs`

Data is pointer to CodecComponentHandle.

Available in Mac OS X v10.0 and later.

Declared in `ImageCompression.h`.

ImageFieldSequenceExtractCombine Values

Constants passed to ImageFieldSequenceExtractCombine.

```
enum {
  evenField1ToEvenFieldOut      = 1 << 0,
  evenField1ToOddFieldOut       = 1 << 1,
  oddField1ToEvenFieldOut       = 1 << 2,
  oddField1ToOddFieldOut        = 1 << 3,
  evenField2ToEvenFieldOut      = 1 << 4,
  evenField2ToOddFieldOut       = 1 << 5,
  oddField2ToEvenFieldOut       = 1 << 6,
  oddField2ToOddFieldOut        = 1 << 7
};
```

QTSetComponentProperty Values

Constants passed to QTSetComponentProperty.

```
enum {
  kComponentPropertyCacheFlagNotPersistent = (1L << 0), /* property metadata should
 not be saved in persistent cache*/
  kComponentPropertyCacheFlagIsDynamic = (1L << 1) /* property metadata should not
 cached at all*/
};
enum {
  kComponentPropertyClassPropertyInfo = 'pnfo', /* property info class */
                                        /* property info property IDs */
  kComponentPropertyInfoList    = 'list', /* array of ComponentPropertyInfo (CFData),
 one for each property */
  kComponentPropertyCacheSeed   = 'seed', /* property cache seed value */
  kComponentPropertyCacheFlags  = 'flgs', /* see kComponentPropertyCache flags */
  kComponentPropertyExtendedInfo = 'meta' /* CFDictionary with extended property
information*/
};
```

Constants

`kComponentPropertyCacheFlagNotPersistent`

Property metadata should not be saved in persistent cache.

Available in Mac OS X v10.3 and later.

Declared in `ImageCompression.h`.

`kComponentPropertyCacheFlagIsDynamic`

Property metadata should not be cached at all.

Available in Mac OS X v10.3 and later.

Declared in `ImageCompression.h`.

`kComponentPropertyClassPropertyInfo`

A QTComponentPropertyInfo structure that defines a property information class. Also 'pnfo'.

Available in Mac OS X v10.3 and later.

Declared in `ImageCompression.h`.

`kComponentPropertyInfoList`

An array of QTComponentPropertyInfo structures, one for each property. Also 'list'.

Available in Mac OS X v10.3 and later.

Declared in `ImageCompression.h`.

`kComponentPropertyCacheSeed`

A component property cache seed value. Also 'seed'.

Available in Mac OS X v10.3 and later.

Declared in `ImageCompression.h`.

`kComponentPropertyCacheFlags`

One of the following two flags: Also 'flgs'.

Available in Mac OS X v10.3 and later.

Declared in `ImageCompression.h`.

kDataHCanRead

Constants grouped with kDataHCanRead.

```
enum {
  kDataHCanRead                = 1L << 0,
  kDataHSpecialRead            = 1L << 1,
  kDataHSpecialReadFile        = 1L << 2,
  kDataHCanWrite               = 1L << 3,
  kDataHSpecialWrite           = 1 << 4,
  kDataHSpecialWriteFile       = 1 << 5,
  kDataHCanStreamingWrite      = 1 << 6,
  kDataHMustCheckDataRef       = 1 << 7
};
```

Constants

`kDataHCanRead`

Indicates that your data handler can read from the volume.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeComponents.h`.

`kDataHSpecialRead`

Indicates that your data handler can read from the volume using a specialized method. For example, your data handler might support access to networked multimedia servers using a special protocol. In that case, your component would set this flag to 1 whenever the volume resides on a supported server.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeComponents.h`.

`kDataHSpecialReadFile`

Reserved for use by Apple.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeComponents.h`.

`kDataHCanWrite`

Indicates that your data handler can write data to the volume. In particular, use this flag to indicate that your data handler's DataHPutData function will work with this volume.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeComponents.h`.

`kDataHSpecialWrite`

Indicates that your data handler can write to the volume using a specialized method. As with the kDataHSpecialRead flag, your data handler would use this flag to indicate that your component can access the volume using specialized support (for example, special network protocols).

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeComponents.h`.

`kDataHCanStreamingWrite`

Indicates that your data handler can support the special write functions for capturing movie data when writing to this volume.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeComponents.h`.

QTVRWrapAndConstrain Values

Constants passed to QTVRWrapAndConstrain.

```
enum {
  kQTVRPan                      = 0,
  kQTVRTilt                     = 1,
  kQTVRFieldOfView              = 2,
  kQTVRViewCenterH              = 4,     /* WrapAndConstrain only*/
  kQTVRViewCenterV              = 5      /* WrapAndConstrain only*/
};
```

Sprite Properties

Constants that represent the properties of sprites.

```
enum {
  kGetSpriteWorldInvalidRegionAndLeaveIntact = -1L,
  kGetSpriteWorldInvalidRegionAndThenSetEmpty = -2L
};
enum {
  kKeyFrameAndSingleOverride    = 1L << 1,
  kKeyFrameAndAllOverrides      = 1L << 2
};
enum {
  kNoQTIdleEvents               = -1
};
enum {
  kOnlyDrawToSpriteWorld        = 1L << 0,
  kSpriteWorldPreflight         = 1L << 1
};
enum {
  kScaleSpritesToScaleWorld     = 1L << 1,
  kSpriteWorldHighQuality       = 1L << 2,
  kSpriteWorldDontAutoInvalidate = 1L << 3,
  kSpriteWorldInvisible         = 1L << 4,
  kSpriteWorldDirtyInsteadOfFlush = 1L << 5
};
enum {
  kSpritePropertyMatrix         = 1,
  kSpritePropertyImageDescription = 2,
  kSpritePropertyImageDataPtr   = 3,
  kSpritePropertyVisible        = 4,
  kSpritePropertyLayer          = 5,
  kSpritePropertyGraphicsMode   = 6,
  kSpritePropertyImageDataSize  = 7,
  kSpritePropertyActionHandlingSpriteID = 8,
  kSpritePropertyCanBeHitTested = 9,
  kSpritePropertyImageIndex     = 100,
  kSpriteTrackPropertyBackgroundColor = 101,
  kSpriteTrackPropertyOffscreenBitDepth = 102,
  kSpriteTrackPropertySampleFormat = 103,
  kSpriteTrackPropertyScaleSpritesToScaleWorld = 104,
  kSpriteTrackPropertyHasActions = 105,
  kSpriteTrackPropertyVisible   = 106,
  kSpriteTrackPropertyQTIdleEventsFrequency = 107,
  kSpriteTrackPropertyAllSpritesHitTestingMode = 108,
  kSpriteTrackPropertyPreferredDepthInterpretationMode = 109,
  kSpriteImagePropertyRegistrationPoint = 1000,
  kSpriteImagePropertyGroupID   = 1001
};
```

QTSampleTableGetSampleFlags Values

Constants passed to QTSampleTableGetSampleFlags.

```
enum {
  mediaSampleNotSync            = 1 << 0, /* sample is not a sync sample (eg. is
frame differenced */
  mediaSampleShadowSync         = 1 << 1, /* sample is a shadow sync */
  mediaSampleDroppable          = 1 << 27, /* sample is not required to be decoded
 for later samples to be decoded properly */
  mediaSamplePartialSync        = 1 << 16, /* sample is a partial sync (e.g., I
frame after open GOP) */
  mediaSampleHasRedundantCoding = 1 << 24, /* sample is known to contain redundant
 coding */
  mediaSampleHasNoRedundantCoding = 1 << 25, /* sample is known not to contain
redundant coding */
  mediaSampleIsDependedOnByOthers = 1 << 26, /* one or more other samples depend
upon the decode of this sample */
  mediaSampleIsNotDependedOnByOthers = 1 << 27, /* synonym for mediaSampleDroppable
 */
  mediaSampleDependsOnOthers    = 1 << 28, /* sample's decode depends upon decode
 of other samples */
  mediaSampleDoesNotDependOnOthers = 1 << 29, /* sample's decode does not depend
upon decode of other samples */
  mediaSampleEarlierDisplayTimesAllowed = 1 << 30 /* samples later in decode order
 may have earlier display times */
};
```

Constants

`mediaSampleNotSync`

Returned for frame-differenced video sample data.

Available in Mac OS X v10.0 and later.

Declared in `Movies.h`.

movieFileSpecValid

Constants grouped with movieFileSpecValid.

```
enum {
  pasteInParallel               = 1 << 0,
  showUserSettingsDialog        = 1 << 1,
  movieToFileOnlyExport         = 1 << 2,
  movieFileSpecValid            = 1 << 3
};
```

MovieImportDataRef Values

Constants passed to MovieImportDataRef.

```
enum {
  movieImportCreateTrack        = 1,
  movieImportInParallel         = 2,
  movieImportMustUseTrack       = 4,
  movieImportWithIdle           = 16
};
enum {
  movieImportResultUsedMultipleTracks = 8,
  movieImportResultNeedIdles    = 32,
  movieImportResultComplete     = 64
};
```

Constants

`movieImportResultNeedIdles`

Undocumented

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeComponents.h`.

MovieProgressProc Values

Constants passed to MovieProgressProc.

```
enum {
  movieProgressOpen             = 0,
  movieProgressUpdatePercent    = 1,
  movieProgressClose            = 2
};
enum {
  progressOpFlatten             = 1,
  progressOpInsertTrackSegment  = 2,
  progressOpInsertMovieSegment  = 3,
  progressOpPaste               = 4,
  progressOpAddMovieSelection   = 5,
  progressOpCopy                = 6,
  progressOpCut                 = 7,
  progressOpLoadMovieIntoRam    = 8,
  progressOpLoadTrackIntoRam    = 9,
  progressOpLoadMediaIntoRam    = 10,
  progressOpImportMovie         = 11,
  progressOpExportMovie         = 12
};
```

Constants

`movieProgressOpen`

Indicates the start of a long operation. This is always the first message sent to your function. Your function can use this message to trigger the display of your progress window.

Available in Mac OS X v10.0 and later.

Declared in `Movies.h`.

`movieProgressUpdatePercent`
Passes completion information to your function. The Movie Toolbox repeatedly sends this message to your function. The percentDone parameter indicates the relative completion of the operation. You can use this value to update your progress window.

Available in Mac OS X v10.0 and later.

Declared in `Movies.h`.

`movieProgressClose`
Indicates the end of a long operation. This is always the last message sent to your function. Your function can use this message as an indication to remove its progress window.

Available in Mac OS X v10.0 and later.

Declared in `Movies.h`.

`progressOpFlatten`
Your application has called the FlattenMovie or FlattenMovieData function.

Available in Mac OS X v10.0 and later.

Declared in `Movies.h`.

`progressOpInsertTrackSegment`
Your application has called the InsertTrackSegment function. The Movie Toolbox calls the progress function that is assigned to the movie that contains the destination track.

Available in Mac OS X v10.0 and later.

Declared in `Movies.h`.

`progressOpInsertMovieSegment`
Your application has called the InsertMovieSegment function. The Movie Toolbox calls the progress function that is assigned to the destination movie.

Available in Mac OS X v10.0 and later.

Declared in `Movies.h`.

`progressOpPaste`
Your application has called the PasteMovieSelection function. The Movie Toolbox calls the progress function that is assigned to the destination movie.

Available in Mac OS X v10.0 and later.

Declared in `Movies.h`.

`progressOpAddMovieSelection`
Your application has called the AddMovieSelection function. The Movie Toolbox calls the progress function that is assigned to the destination movie. The Movie Toolbox calls the progress function that is assigned to the destination movie.

Available in Mac OS X v10.0 and later.

Declared in `Movies.h`.

`progressOpCopy`
Your application has called the CopyMovieSelection function The Movie Toolbox calls the progress function that is assigned to the destination movie.

Available in Mac OS X v10.0 and later.

Declared in `Movies.h`.

`progressOpCut`
Your application has called the CutMovieSelection function. The Movie Toolbox calls the progress function that is assigned to the destination movie.

Available in Mac OS X v10.0 and later.

Declared in `Movies.h`.

`progressOpLoadMovieIntoRam`
Your application has called the LoadMovieIntoRam function. The Movie Toolbox calls the progress function that is assigned to the destination movie.

Available in Mac OS X v10.0 and later.

Declared in `Movies.h`.

`progressOpLoadTrackIntoRam`
Your application has called the LoadTrackIntoRam function. The Movie Toolbox calls the progress function that is assigned to the destination track.

Available in Mac OS X v10.0 and later.

Declared in `Movies.h`.

`progressOpLoadMediaIntoRam`
Your application has called the LoadMediaIntoRam function. The Movie Toolbox calls the progress function that is assigned to the destination media.

Available in Mac OS X v10.0 and later.

Declared in `Movies.h`.

`progressOpImportMovie`
Your application has called the ConvertFileToMovieFile function. The Movie Toolbox calls the progress function that is associated with the destination movie file. This flag is also used, as appropriate, for the PasteHandleIntoMovie functions.

Available in Mac OS X v10.0 and later.

Declared in `Movies.h`.

New Movie Properties

Constants that represent the properties of new movies.

```
enum {
  newMovieActive                = 1 << 0,
  newMovieDontResolveDataRefs   = 1 << 1,
  newMovieDontAskUnresolvedDataRefs = 1 << 2,
  newMovieDontAutoAlternates    = 1 << 3,
  newMovieDontUpdateForeBackPointers = 1 << 4,
  newMovieDontAutoUpdateClock   = 1 << 5,
  newMovieAsyncOK               = 1 << 8,
  newMovieIdleImportOK          = 1 << 10,
  newMovieDontInteractWithUser  = 1 << 11
};
```

NewMovieController Values

Constants passed to NewMovieController.

```
enum {
  mcTopLeftMovie                = 1 << 0, /* usually centered */
  mcScaleMovieToFit             = 1 << 1, /* usually only scales down */
  mcWithBadge                   = 1 << 2, /* give me a badge */
  mcNotVisible                  = 1 << 3, /* don't show controller */
  mcWithFrame                   = 1 << 4 /* gimme a frame */
};
```

QuickTime Preferences Dialog Options

Constants that represent options for QuickTime preference dialogs.

```
enum {
  pdActionConfirmDialog         = 1,    /* no param*/
  pdActionSetAppleMenu          = 2,    /* param is MenuRef*/
  pdActionSetEditMenu           = 3,    /* param is MenuRef*/
  pdActionGetDialogValues       = 4,    /* param is QTAtomContainer*/
  pdActionSetPreviewUserItem    = 5,    /* param is long*/
  pdActionSetPreviewPicture     = 6,    /* param is QTParamPreviewPtr;*/
  pdActionSetColorPickerEventProc = 7,  /* param is UserEventUPP*/
  pdActionSetDialogTitle        = 8,    /* param is StringPtr */
  pdActionGetSubPanelMenu       = 9,    /* param is MenuRef* */
  pdActionActivateSubPanel      = 10,   /* param is long */
  pdActionConductStopAlert      = 11,   /* param is StringPtr */
  pdActionModelessCallback      = 12,   /* param is QTParamDialogEventPtr */
  pdActionFetchPreview          = 13,   /* param is QTParamFetchPreviewPtr */
  pdActionSetDialogSettings     = 14,   /* param is QTAtomContainer */
  pdActionGetDialogSettings     = 15,   /* param is QTAtomContainer */
  pdActionGetNextSample         = 16,   /* param is QTAtomContainer with effect
sample to change - createdDialog may be NIL */
  pdActionGetPreviousSample     = 17,   /* param is QTAtomContainer with effect
sample to change - createdDialog may be NIL */
  pdActionCompactSample         = 18,   /* param is QTAtomContainer with effect
sample to compact, - createdDialog may be NIL */
  pdActionSetEditCallout        = 19,   /* param is QTParamPreviewCalloutPtr, can
 be NIL */
  pdActionSetSampleTime         = 20,   /* param is QTParamSampleTimePtr, can be
NIL */
  pdActionDoEditCommand         = 21,   /* param is long with menu command (ie,
mcMenuCut etc) */
  pdActionGetSubPanelMenuValue  = 22,   /* param is long and returns current
sub-panel value selected by the effect */
                                      /* Action codes and typedefs used for custom
 controls within effects */
  pdActionCustomNewControl      = 23,   /* param is QTCustomControlNewPtr */
  pdActionCustomDisposeControl  = 24,   /* param is QTCustomControlNewPtr */
 pdActionCustomPositionControl = 25,   /* param is QTCustomControlPositionControlPtr
 */
 pdActionCustomShowHideControl = 26,   /* param is QTCustomControlShowHideControlPtr
 */
  pdActionCustomHandleEvent     = 27,   /* param is QTCustomControlHandleEventPtr
 */
  pdActionCustomSetFocus        = 28,   /* param is QTCustomControlSetFocusPtr */
  pdActionCustomSetEditMenu     = 29,   /* param is QTCustomControlSetEditMenuPtr
 */
  pdActionCustomSetPreviewPicture = 30, /* param is
QTCustomControlSetPreviewPicturePtr */
 pdActionCustomSetEditCallout  = 31,   /* param is QTCustomControlSetEditCalloutPtr
 */
 pdActionCustomGetEnableValue  = 32,   /* param is QTCustomControlGetEnableValuePtr
 */
 pdActionCustomSetSampleTime   = 33,   /* param is QTCustomControlSetSampleTimePtr
 */
  pdActionCustomGetValue        = 34,   /* param is QTCustomControlGetValue */
  pdActionCustomDoEditCommand   = 35,   /* param is QTCustomControlDoEditCommand
*/
                                        /* more actions for the dialog */
  pdActionRunInEventLoop        = 36,   /* param is QTEventLoopDescriptionPtr - OS
 X only*/
  pdActionConvertSettingsToXML  = 37,   /* param is QTAtomContainer* inbound,
Handle* outbound contains the XML - createdDialog may be NIL */
  pdActionConvertSettingsToXMLWithComments = 38, /* param is QTAtomContainer*
inbound, Handle* outbound contains the XML with comments - createdDialog may be
NIL */
  pdActionConvertSettingsToText = 39,   /* param is QTAtomContainer* inbound,
Handle* outbound contains human readable text - createdDialog may be NIL */
  pdActionConvertXMLToSettings  = 40,   /* param is Handle* inbound, QTAtomContainer*
 outbound contains parameters - createdDialog may be NIL */
  pdActionSetPropertyComponent  = 41    /* param is QTParamComponentPropertyPtr */
};
enum {
  pdOptionsCollectOneValue      = 0x00000001, /* should collect a single value
only*/
  pdOptionsAllowOptionalInterpolations = 0x00000002, /* non-novice interpolation
options are shown */
  pdOptionsModalDialogBox       = 0x00000004, /* dialog box should be modal */
  pdOptionsEditCurrentEffectOnly = 0x00000008, /* List of effects will not be shown
 */
  pdOptionsHidePreview          = 0x00000010, /* Preview item will not be shown */
  pdOptionsDisplayAsSheet       = 0x00000020 /* Dialog will be used as a sheet (on
 platforms that support it) */
};
enum {
  pdSampleTimeDisplayOptionsNone = 0x00000000
};
```

Constants

`pdActionModelessCallback`

Parameter is QTParamDialogEventPtr.

Available in Mac OS X v10.0 and later.

Declared in `Movies.h`.

`pdActionFetchPreview`

Parameter is QTParamFetchPreviewPtr.

Available in Mac OS X v10.0 and later.

Declared in `Movies.h`.

`pdActionSetDialogSettings`

Parameter is QTAtomContainer.

Available in Mac OS X v10.2 and later.

Declared in `Movies.h`.

`pdActionGetDialogSettings`

Parameter is QTAtomContainer.

Available in Mac OS X v10.2 and later.

Declared in `Movies.h`.

`pdActionGetNextSample`

Parameter is QTAtomContainer with effect sample to change - createdDialog may be NIL.

Available in Mac OS X v10.2 and later.

Declared in `Movies.h`.

`pdActionGetPreviousSample`
: Parameter is QTAtomContainer with effect sample to change - createdDialog may be NIL.

  Available in Mac OS X v10.2 and later.

  Declared in `Movies.h`.

`pdActionCompactSample`
: Parameter is QTAtomContainer with effect sample to compact, - createdDialog may be NIL.

  Available in Mac OS X v10.2 and later.

  Declared in `Movies.h`.

`pdActionSetEditCallout`
: Parameter is QTParamPreviewCalloutPtr, can be NIL.

  Available in Mac OS X v10.2 and later.

  Declared in `Movies.h`.

`pdActionSetSampleTime`
: Parameter is QTParamSampleTimePtr, can be NIL.

  Available in Mac OS X v10.2 and later.

  Declared in `Movies.h`.

`pdActionDoEditCommand`
: Parameter is long with menu command (that is, mcMenuCut etc).

  Available in Mac OS X v10.2 and later.

  Declared in `Movies.h`.

`pdActionGetSubPanelMenuValue`
: Parameter is long and returns current sub-panel value selected by the effect.

  Available in Mac OS X v10.2 and later.

  Declared in `Movies.h`.

`pdActionCustomNewControl`
: Parameter is QTCustomControlNewPtr.

  Available in Mac OS X v10.2 and later.

  Declared in `Movies.h`.

`pdActionCustomDisposeControl`
: Parameter is QTCustomControlNewPtr.

  Available in Mac OS X v10.2 and later.

  Declared in `Movies.h`.

`pdActionCustomPositionControl`
: Parameter is QTCustomControlPositionControlPtr.

  Available in Mac OS X v10.2 and later.

  Declared in `Movies.h`.

`pdActionCustomShowHideControl`
: Parameter is QTCustomControlShowHideControlPtr.

  Available in Mac OS X v10.2 and later.

  Declared in `Movies.h`.

`pdActionCustomHandleEvent`
Parameter is QTCustomControlHandleEventPtr.
Available in Mac OS X v10.2 and later.
Declared in `Movies.h`.

`pdActionCustomSetFocus`
Parameter is QTCustomControlSetFocusPtr.
Available in Mac OS X v10.2 and later.
Declared in `Movies.h`.

`pdActionCustomSetEditMenu`
Parameter is QTCustomControlSetEditMenuPtr.
Available in Mac OS X v10.2 and later.
Declared in `Movies.h`.

`pdActionCustomSetPreviewPicture`
Parameter is QTCustomControlSetPreviewPicturePtr.
Available in Mac OS X v10.2 and later.
Declared in `Movies.h`.

`pdActionCustomSetEditCallout`
Parameter is QTCustomControlSetEditCalloutPtr.
Available in Mac OS X v10.2 and later.
Declared in `Movies.h`.

`pdActionCustomGetEnableValue`
Parameter is QTCustomControlGetEnableValuePtr.
Available in Mac OS X v10.2 and later.
Declared in `Movies.h`.

`pdActionCustomSetSampleTime`
Parameter is QTCustomControlSetSampleTimePtr.
Available in Mac OS X v10.2 and later.
Declared in `Movies.h`.

`pdActionCustomGetValue`
Parameter is QTCustomControlGetValue.
Available in Mac OS X v10.2 and later.
Declared in `Movies.h`.

`pdActionCustomDoEditCommand`
Parameter is QTCustomControlDoEditCommand.
Available in Mac OS X v10.2 and later.
Declared in `Movies.h`.

`pdActionRunInEventLoop`
Parameter is QTEventLoopDescriptionPtr - OS X only.
Available in Mac OS X v10.3 and later.
Declared in `Movies.h`.

`pdActionConvertSettingsToXML`
: Parameter is QTAtomContainer inbound, Handle outbound contains the XML - createdDialog may be NIL.

  Available in Mac OS X v10.3 and later.

  Declared in `Movies.h`.

`pdActionConvertSettingsToXMLWithComments`
: Parameter is QTAtomContainer inbound, Handle outbound contains the XML with comments - createdDialog may be NIL.

  Available in Mac OS X v10.3 and later.

  Declared in `Movies.h`.

`pdActionConvertSettingsToText`
: Parameter is QTAtomContainer inbound, Handle outbound contains human readable text - createdDialog may be NIL.

  Available in Mac OS X v10.3 and later.

  Declared in `Movies.h`.

`pdActionConvertXMLToSettings`
: Parameter is Handle inbound, QTAtomContainer outbound contains parameters - createdDialog may be NIL.

  Available in Mac OS X v10.3 and later.

  Declared in `Movies.h`.

`pdActionSetPropertyComponent`
: Parameter is QTParamComponentPropertyPtr.

  Available in Mac OS X v10.3 and later.

  Declared in `Movies.h`.

`pdOptionsModalDialogBox`
: Dialog box should be modal.

  Available in Mac OS X v10.0 and later.

  Declared in `Movies.h`.

`pdOptionsEditCurrentEffectOnly`
: List of effects will not be shown.

  Available in Mac OS X v10.2 and later.

  Declared in `Movies.h`.

`pdOptionsHidePreview`
: Preview item will not be shown.

  Available in Mac OS X v10.2 and later.

  Declared in `Movies.h`.

Standard Compression Constants

Constants that represent constants for Standard Compression.

```
enum {
  /*
   * Indicates the client is ready to use the ICM compression session
   * API to perform compression operations. StdCompression disables
   * frame reordering and multi pass encoding if this flag is cleared.
   */
  scAllowEncodingWithCompressionSession = 1L << 8,
  /*
   * Indicates the client does not want the user to change the frame
   * reordering setting.
   */
  scDisableFrameReorderingItem  = 1L << 9,
  /*
   * Indicates the client does not want the user to change the multi
   * pass encoding setting
   */
  scDisableMultiPassEncodingItem = 1L << 10
};
enum {
  /*
   * Specifies if frame reordering can occur in encoding.
   */
  scVideoAllowFrameReorderingType = 'bfra', /* pointer to Boolean*/
  /*
   * The settings to control multi pass encoding.
   */
  scVideoMultiPassEncodingSettingsType = 'mpes' /* pointer to
SCVideoMutiPassEncodingSettings struct*/
};
enum {
  scListEveryCodec              = 1L << 1,
  scAllowZeroFrameRate          = 1L << 2,
  scAllowZeroKeyFrameRate       = 1L << 3,
  scShowBestDepth               = 1L << 4,
  scUseMovableModal             = 1L << 5,
  scDisableFrameRateItem        = 1L << 6,
  scShowDataRateAsKilobits      = 1L << 7
};
enum {
  scOKItem                      = 1,
  scCancelItem                  = 2,
  scCustomItem                  = 3
};
enum {
  scPositionRect                = 2,
  scPositionDialog              = 3,
  scSetTestImagePictHandle      = 4,
  scSetTestImagePictFile        = 5,
  scSetTestImagePixMap          = 6,
  scGetBestDeviceRect           = 7,
  scRequestImageSettings        = 10,
  scCompressImage               = 11,
  scCompressPicture             = 12,
  scCompressPictureFile         = 13,
  scRequestSequenceSettings     = 14,
  scCompressSequenceBegin       = 15,
  scCompressSequenceFrame       = 16,
  scCompressSequenceEnd         = 17,
  scDefaultPictHandleSettings   = 18,
  scDefaultPictFileSettings     = 19,
  scDefaultPixMapSettings       = 20,
  scGetInfo                     = 21,
  scSetInfo                     = 22,
  scNewGWorld                   = 23
};
enum {
  scPreferCropping              = 1 << 0,
  scPreferScaling               = 1 << 1,
  scPreferScalingAndCropping    = scPreferScaling | scPreferCropping,
  scDontDetermineSettingsFromTestImage = 1 << 2
};
enum {
  scSpatialSettingsType         = 'sptl', /* pointer to SCSpatialSettings struct*/
  scTemporalSettingsType        = 'tprl', /* pointer to SCTemporalSettings struct*/
  scDataRateSettingsType        = 'drat', /* pointer to SCDataRateSettings struct*/
  scColorTableType              = 'clut', /* pointer to CTabHandle*/
  scProgressProcType            = 'prog', /* pointer to ProgressRecord struct*/
  scExtendedProcsType           = 'xprc', /* pointer to SCExtendedProcs struct*/
  scPreferenceFlagsType         = 'pref', /* pointer to long*/
  scSettingsStateType           = 'ssta', /* pointer to Handle*/
  scSequenceIDType              = 'sequ', /* pointer to ImageSequence*/
  scWindowPositionType          = 'wndw', /* pointer to Point*/
  scCodecFlagsType              = 'cflg', /* pointer to CodecFlags*/
  scCodecSettingsType           = 'cdec', /* pointer to Handle*/
  scForceKeyValueType           = 'ksim', /* pointer to long*/
  scCompressionListType         = 'ctyl', /* pointer to OSType Handle*/
  scCodecManufacturerType       = 'cmfr', /* pointer to OSType*/
  scAvailableCompressionListType = 'avai', /* pointer to OSType Handle*/
  scWindowOptionsType           = 'shee', /* pointer to SCWindowSettings struct*/
  scSoundVBRCompressionOK       = 'cvbr', /* pointer to Boolean*/
  scSoundSampleRateChangeOK     = 'rcok', /* pointer to Boolean*/
  scSoundCompressionType        = 'ssct', /* pointer to OSType*/
  scSoundSampleRateType         = 'ssrt', /* pointer to UnsignedFixed*/
  scSoundInputSampleRateType    = 'ssir', /* pointer to UnsignedFixed*/
  scSoundSampleSizeType         = 'ssss', /* pointer to short*/
  scSoundChannelCountType       = 'sscc' /* pointer to short*/
};
enum {
  scTestImageWidth              = 80,
  scTestImageHeight             = 80
};
enum {
  scUserCancelled               = 1
};
enum {
  scWindowRefKindCarbon         = 'carb' /* WindowRef*/
};
```

Constants

`scVideoAllowFrameReorderingType`

Pointer to Boolean.

Available in Mac OS X v10.3 and later.

Declared in `QuickTimeComponents.h`.

`scSpatialSettingsType`
: A video track's SCSpatialSettings structure.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`scTemporalSettingsType`
: A video track's SCTemporalSettings structure.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`scDataRateSettingsType`
: A video track's SCDataRateSettings structure.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`scCodecSettingsType`
: Pointer to Handle.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`scForceKeyValueType`
: Pointer to long.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`scCodecManufacturerType`
: Pointer to OSType.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`scAvailableCompressionListType`
: Pointer to OSType Handle.

  Available in Mac OS X v10.2 and later.

  Declared in `QuickTimeComponents.h`.

`scWindowOptionsType`
: Pointer to SCWindowSettings struct.

  Available in Mac OS X v10.3 and later.

  Declared in `QuickTimeComponents.h`.

`scSoundVBRCompressionOK`
: Pointer to Boolean.

  Available in Mac OS X v10.2 and later.

  Declared in `QuickTimeComponents.h`.

`scSoundSampleRateChangeOK`
: Pointer to Boolean.

  Available in Mac OS X v10.2 and later.

  Declared in `QuickTimeComponents.h`.

`scSoundCompressionType`
A sound track's compression type constant; see Codec Identifiers.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeComponents.h`.

`scSoundSampleRateType`
An UnsignedFixed value that represents a sound track's sampling rate.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeComponents.h`.

`scSoundInputSampleRateType`
Pointer to UnsignedFixed.

Available in Mac OS X v10.2 and later.

Declared in `QuickTimeComponents.h`.

`scSoundSampleSizeType`
A short integer that represents a sound track's sample size.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeComponents.h`.

`scSoundChannelCountType`
A short integer that represents a sound track's channel count.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeComponents.h`.

SGPanelGetDITLForSize Values

Constants passed to SGPanelGetDITLForSize.

```
enum {
  kSGSmallestDITLSize           = -1,   /* requestedSize h and v set to this to
retrieve small size*/
  kSGLargestDITLSize            = -2    /* requestedSize h and v set to this to
retrieve large size*/
};
```

Media Identifiers

Identify media types in QuickTime.

```
enum {
  VideoMediaType                = 'vide',
  SoundMediaType                = 'soun',
  TextMediaType                 = 'text',
  BaseMediaType                 = 'gnrc',
  MPEGMediaType                 = 'MPEG',
  MusicMediaType                = 'musi',
  TimeCodeMediaType             = 'tmcd',
  SpriteMediaType               = 'sprt',
  FlashMediaType                = 'flsh',
  MovieMediaType                = 'moov',
  TweenMediaType                = 'twen',
  ThreeDeeMediaType             = 'qd3d',
  SkinMediaType                 = 'skin',
  HandleDataHandlerSubType      = 'hndl',
  PointerDataHandlerSubType     = 'ptr ',
  NullDataHandlerSubType        = 'null',
  ResourceDataHandlerSubType    = 'rsrc',
  URLDataHandlerSubType         = 'url ',
  AliasDataHandlerSubType       = 'alis',
  WiredActionHandlerType        = 'wire'
};
```

Constants

`SoundMediaType`

Sound channel.

Available in Mac OS X v10.0 and later.

Declared in `Movies.h`.

`TextMediaType`

Text media.

Available in Mac OS X v10.0 and later.

Declared in `Movies.h`.

SpriteWorldHitTest Values

Constants passed to SpriteWorldHitTest.

```
enum {
  spriteHitTestBounds           = 1L << 0, /*    point must only be within sprite's
 bounding box*/
  spriteHitTestImage            = 1L << 1, /*  point must be within the shape of
the sprite's image*/
  spriteHitTestInvisibleSprites = 1L << 2, /*  invisible sprites may be hit tested*/
  spriteHitTestIsClick          = 1L << 3, /*  for codecs that want mouse events*/
  spriteHitTestLocInDisplayCoordinates = 1L << 4, /*    set if you want to pass a
 display coordiate point to SpriteHitTest*/
  spriteHitTestTreatAllSpritesAsHitTestable = 1L << 5 /* set if you want to override
 each sprites hittestable property as true*/
};
```

Text Properties

Constants that represent the properties of text.

```
enum {
                                        /* set property parameter / get property
parameter*/
  kTextTextHandle               = 1,    /* Handle / preallocated Handle*/
  kTextTextPtr                  = 2,    /* Pointer*/
  kTextTEStyle                  = 3,    /* TextStyle * / TextStyle **/
  kTextSelection                = 4,    /* long [2] / long [2]*/
  kTextBackColor                = 5,    /* RGBColor * / RGBColor **/
  kTextForeColor                = 6,    /* RGBColor * / RGBColor **/
  kTextFace                     = 7,    /* long / long **/
  kTextFont                     = 8,    /* long / long **/
  kTextSize                     = 9,    /* long / long **/
  kTextAlignment                = 10,   /* short * / short **/
  kTextHilite                   = 11,   /* hiliteRecord * / hiliteRecord **/
  kTextDropShadow               = 12,   /* dropShadowRecord * / dropShadowRecord
**/
  kTextDisplayFlags             = 13,   /* long / long **/
  kTextScroll                   = 14,   /* TimeValue * / TimeValue **/
  kTextRelativeScroll           = 15,   /* Point **/
  kTextHyperTextFace            = 16,   /* hyperTextSetFace * / hyperTextSetFace
**/
  kTextHyperTextColor           = 17,   /* hyperTextSetColor * / hyperTextSetColor
 **/
  kTextKeyEntry                 = 18,   /* short*/
  kTextMouseDown                = 19,   /* Point **/
  kTextTextBox                  = 20,   /* Rect * / Rect **/
  kTextEditState                = 21,   /* short / short **/
  kTextLength                   = 22    /*       / long **/
};
enum {
  dfDontDisplay                 = 1 << 0, /* Don't display the text*/
  dfDontAutoScale              = 1 << 1, /* Don't scale text as track bounds grows
 or shrinks*/
  dfClipToTextBox               = 1 << 2, /* Clip update to the textbox*/
  dfUseMovieBGColor             = 1 << 3, /* Set text background to movie's
background color*/
  dfShrinkTextBoxToFit         = 1 << 4, /* Compute minimum box to fit the sample*/
  dfScrollIn                    = 1 << 5, /* Scroll text in until last of text is
 in view */
  dfScrollOut                   = 1 << 6, /* Scroll text out until last of text is
 gone (if both set, scroll in then out)*/
  dfHorizScroll                 = 1 << 7, /* Scroll text horizontally (otherwise
it's vertical)*/
  dfReverseScroll               = 1 << 8, /* vert: scroll down rather than up;
horiz: scroll backwards (justfication dependent)*/
  dfContinuousScroll            = 1 << 9, /* new samples cause previous samples to
 scroll out */
  dfFlowHoriz                   = 1 << 10, /* horiz scroll text flows in textbox
rather than extend to right */
  dfContinuousKaraoke          = 1 << 11, /* ignore begin offset, hilite everything
 up to the end offset(karaoke)*/
  dfDropShadow                  = 1 << 12, /* display text with a drop shadow */
  dfAntiAlias                  = 1 << 13, /* attempt to display text anti aliased*/
  dfKeyedText                   = 1 << 14, /* key the text over background*/
  dfInverseHilite               = 1 << 15, /* Use inverse hiliting rather than
using hilite color*/
  dfTextColorHilite            = 1 << 16 /* changes text color in place of hiliting.
 */
};
```

Constants

`kTextSelection`
: Long [2] long [2].

  Available in Mac OS X v10.0 and later.

  Declared in `Movies.h`.

`kTextScroll`
: The text scroll position.

  Available in Mac OS X v10.0 and later.

  Declared in `Movies.h`.

`kTextRelativeScroll`
: Point.

  Available in Mac OS X v10.0 and later.

  Declared in `Movies.h`.

`kTextHyperTextFace`
: HyperTextSetFace hyperTextSetFace.

  Available in Mac OS X v10.0 and later.

  Declared in `Movies.h`.

`kTextHyperTextColor`
: HyperTextSetColor hyperTextSetColor.

  Available in Mac OS X v10.0 and later.

  Declared in `Movies.h`.

`kTextKeyEntry`
: Short.

  Available in Mac OS X v10.0 and later.

  Declared in `Movies.h`.

`kTextMouseDown`
: Point.

  Available in Mac OS X v10.0 and later.

  Declared in `Movies.h`.

`kTextTextBox`
: Rect Rect.

  Available in Mac OS X v10.0 and later.

  Declared in `Movies.h`.

`kTextEditState`
: Short short.

  Available in Mac OS X v10.0 and later.

  Declared in `Movies.h`.

`kTextLength`
: Long.

  Available in Mac OS X v10.0 and later.

  Declared in `Movies.h`.

`dfDontDisplay`
Does not display the specified sample.

Available in Mac OS X v10.0 and later.

Declared in `Movies.h`.

`dfDontAutoScale`
Does not scale the text if the track bounds increase.

Available in Mac OS X v10.0 and later.

Declared in `Movies.h`.

`dfClipToTextBox`
Clips to just the text box. This is useful if the text overlays the video.

Available in Mac OS X v10.0 and later.

Declared in `Movies.h`.

`dfUseMovieBGColor`
Set text background to movie's background color.

Available in Mac OS X v10.0 and later.

Declared in `Movies.h`.

`dfShrinkTextBoxToFit`
Recalculates size of the textBox parameter to just fit the given text and stores this rectangle with the text data.

Available in Mac OS X v10.0 and later.

Declared in `Movies.h`.

`dfScrollIn`
Scrolls the text in until the last of the text is in view. This flag is associated with the scrollDelay parameter.

Available in Mac OS X v10.0 and later.

Declared in `Movies.h`.

`dfScrollOut`
Scrolls text out until the last of the text is out of view. This flag is associated with the scrollDelay parameter. If both dfScrollIn and dfScrollOut are set, the text is scrolled in, then out.

Available in Mac OS X v10.0 and later.

Declared in `Movies.h`.

`dfHorizScroll`
Scrolls a single line of text horizontally. If the dfHorizScroll flag is not set, then the scrolling is vertical.

Available in Mac OS X v10.0 and later.

Declared in `Movies.h`.

`dfReverseScroll`
If set, scrolls vertically down, rather than up. If not set, horizontal scrolling proceeds toward the left rather than toward the right.

Available in Mac OS X v10.0 and later.

Declared in `Movies.h`.

ToneDescription Values

Constants passed to ToneDescription.

```
enum {
  kSoftSynthComponentSubType    = 'ss  ',
  kGMSynthComponentSubType      = 'gm  '
};
```

Constants

`kSoftSynthComponentSubType`

Software synthesizer; value is 'ss '.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeMusic.h`.

Arithmetic and Logical Operator IDs

Constants that identify arithmetic and logical operations.

```
kOperatorAdd                  = 'add '
kOperatorSubtract             = 'sub '
kOperatorMultiply             = 'mult'
kOperatorDivide               = 'div '
kOperatorOr                   = 'or  '
kOperatorAnd                  = 'and '
kOperatorNot                  = 'not '
kOperatorLessThan             = '<   '
kOperatorLessThanEqualTo      = '<=  '
kOperatorEqualTo              = '=   '
kOperatorNotEqualTo           = '!=  '
kOperatorGreaterThan          = '>   '
kOperatorGreaterThanEqualTo   = '>=  '
kOperatorModulo               = 'mod '
kOperatorIntegerDivide        = 'idiv'
kOperatorAbsoluteValue        = 'abs '
kOperatorNegate               = 'neg '
```

Codec Identifiers

Identify codec components and data types in QuickTime.

```
kAnimationCodecType          ='rle '
kAVRJPEGCodecType            ='avr '
kBaseCodecType               ='base'
kBMPCodecType                ='WRLE'
kCinepakCodecType            ='cvid'
kCloudCodecType              ='clou'
kCMYKCodecType               ='cmyk'
kComponentVideoCodecType     ='yuv2'
kComponentVideoSigned        ='yuvu'
kComponentVideoUnsigned      ='yuvs'
kDVCNTSCCodecType            ='dvc '
kDVCPALCodecType             ='dvcp'
kDVCProNTSCCodecType         ='dvpn'
kDVCProPALCodecType          ='dvpp'
kFireCodecType               ='fire'
kFLCCodecType                ='flic'
k48RGBCodecType              ='b48r'
kGIFCodecType                ='gif '
kGraphicsCodecType           ='smc '
kH261CodecType               ='h261'
kH263CodecType               ='h263'
kIndeo4CodecType             ='IV41'
kJPEGCodecType               ='jpeg'
kMacPaintCodecType           ='PNTG'
kMicrosoftVideo1CodecType    ='msvc'
kMotionJPEGACodecType        ='mjpa'
kMotionJPEGBCodecType        ='mjpb'
kMpegYUV420CodecType         ='myuv'
kOpenDMLJPEGCodecType        ='dmb1'
kPhotoCDCodecType            ='kpcd'
kPlanarRGBCodecType          ='8BPS'
kPNGCodecType                ='png '
kQuickDrawCodecType          ='qdrw'
kQuickDrawGXCodecType        ='qdgx'
kRawCodecType                ='raw '
kSGICodecType                ='.SGI'
k16GrayCodecType             ='b16g'
k64ARGBCodecType             ='b64a'
kSorensonCodecType           ='SVQ1'
kSorensonYUV9CodecType       ='syv9'
kTargaCodecType              ='tga '
k32AlphaGrayCodecType        ='b32a'
kTIFFCodecType               ='tiff'
kVectorCodecType             ='path'
kVideoCodecType              ='rpza'
kWaterRippleCodecType        ='ripl'
kWindowsRawCodecType         ='WRAW'
kYUV420CodecType             ='y420
```

Discussion
All codec components of the same type provide the same kinds of services and support a common application programming interface.

Codec Properties

Constants that represent the properties of codecs.

```
codecImageBufferIsInPCIMemory = 1L << 5
codecSupportsOutOfOrderDisplayTimes = 1L << 8
codecSupportsScheduledBackwardsPlaybackWithDifferenceFrames = 1L << 9
codecConditionNewMask         = 1L << 6
codecInfoResourceType         = 'cdci'
codecInterfaceVersion         = 2
codecSuggestedBufferSentinel  = 'sent'
codecMinimumDataSize          = 32768L
```

`codecImageBufferIsInPCIMemory`
Codec image buffer is across a PCI bus; byte writes are bad.

Available in Mac OS X v10.0 and later.

Declared in `ImageCodec.h`.

`codecSupportsOutOfOrderDisplayTimes`
Codec supports frames queued in one order for display in a different order, for example IPB content.

Available in Mac OS X v10.3 and later.

Declared in `ImageCodec.h`.

`codecSupportsScheduledBackwardsPlaybackWithDifferenceFrames`
Codec can use additional buffers to minimize redecoding during backwards playback.

Available in Mac OS X v10.3 and later.

Declared in `ImageCodec.h`.

`codecInfoResourceType`
Codec info resource type.

Available in Mac OS X v10.0 and later.

Declared in `ImageCodec.h`.

`codecInterfaceVersion`
High word returned in component GetVersion.

Available in Mac OS X v10.0 and later.

Declared in `ImageCodec.h`.

`codecSuggestedBufferSentinel`
Codec public resource containing suggested data pattern to put past end of data buffer.

Available in Mac OS X v10.2 and later.

Declared in `ImageCodec.h`.

Codec Type Constants

Constants that represent codec types.

```
kDVCPro50NTSCCodecType          = 'dv5n'
kDVCPro50PALCodecType           = 'dv5p'
kDVCPro100NTSCCodecType         = 'dv1n'
kDVCPro100PALCodecType          = 'dv1p'
kDVCPROHD720pCodecType          = 'dvhp'
kDVCPROHD1080i60CodecType       = 'dvh6'
kDVCPROHD1080i50CodecType       = 'dvh5'
kSorenson3CodecType             = 'SVQ3'
kMPEG4VisualCodecType           = 'mp4v'
k422YpCbCr8CodecType            = '2vuy'
k444YpCbCr8CodecType            = 'v308'
k4444YpCbCrA8CodecType          = 'v408'
k422YpCbCr16CodecType           = 'v216'
k422YpCbCr10CodecType           = 'v210'
k444YpCbCr10CodecType           = 'v410'
k4444YpCbCrA8RCodecType         = 'r408'
kJPEG2000CodecType              = 'mjp2'
kPixletCodecType                = 'pxlt'
kH264CodecType                  = 'avc1'
```

`kSorenson3CodecType`
: Available in QuickTime 5 and later.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCompression.h`.

`k422YpCbCr8CodecType`
: Component Y'CbCr 8-bit 4:2:2.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCompression.h`.

`k444YpCbCr8CodecType`
: Component Y'CbCr 8-bit 4:4:4.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCompression.h`.

`k4444YpCbCrA8CodecType`
: Component Y'CbCrA 8-bit 4:4:4:4.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCompression.h`.

`k422YpCbCr16CodecType`
: Component Y'CbCr 10,12,14,16-bit 4:2:2.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCompression.h`.

`k422YpCbCr10CodecType`
: Component Y'CbCr 10-bit 4:2:2.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCompression.h`.

`k444YpCbCr10CodecType`
: Component Y'CbCr 10-bit 4:4:4.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCompression.h`.

`k4444YpCbCrA8RCodecType`
: Component Y'CbCrA 8-bit 4:4:4:4, rendering format. full range alpha, zero biased YUV.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCompression.h`.

Color Constants

Identify default colors for a graphics importer component.

```
blackColor      =33
blueColor       =409
cyanColor       =273
greenColor      =341
magentaColor    =137
redColor        =205
whiteColor      =30
yellowColor     =69
```

Color Modes

Constants that represent color modes.

```
useColorMatching              = 4
graphicsModePreWhiteAlpha     = 257
graphicsModePreBlackAlpha     = 258
graphicsModeComposition       = 259
graphicsModePreMulColorAlpha  = 261
graphicsModePerComponentAlpha = 272
kQTAlphaMode                  = 'almo'
kQTAlphaModePreMulColor       = 'almp'
```

`kQTAlphaMode`
: UInt32; for example, graphicsModeStraightAlpha or graphicsModePreBlackAlpha.

  Available in Mac OS X v10.1 and later.

  Declared in `ImageCompression.h`.

`kQTAlphaModePreMulColor`
: RGBColor; used if kQTAlphaMode is graphicsModePreMulColorAlpha.

  Available in Mac OS X v10.1 and later.

  Declared in `ImageCompression.h`.

Component Call Selectors

Constants that represent selectors for component calls.

```
kClockGetTimeSelect                         = 0x0001
kClockNewCallBackSelect                     = 0x0002
kClockDisposeCallBackSelect                 = 0x0003
kClockCallMeWhenSelect                      = 0x0004
kClockCancelCallBackSelect                  = 0x0005
kClockRateChangedSelect                     = 0x0006
kClockTimeChangedSelect                     = 0x0007
kClockSetTimeBaseSelect                     = 0x0008
kClockStartStopChangedSelect                = 0x0009
kClockGetRateSelect                         = 0x000A
kClockGetTimesForRateChangeSelect           = 0x000B
kClockGetRateChangeConstraintsSelect        = 0x000C
kSCGetCompressionExtendedSelect             = 0x0001
kSCPositionRectSelect                       = 0x0002
kSCPositionDialogSelect                     = 0x0003
kSCSetTestImagePictHandleSelect             = 0x0004
kSCSetTestImagePictFileSelect               = 0x0005
kSCSetTestImagePixMapSelect                 = 0x0006
kSCGetBestDeviceRectSelect                  = 0x0007
kSCRequestImageSettingsSelect               = 0x000A
kSCCompressImageSelect                      = 0x000B
kSCCompressPictureSelect                    = 0x000C
kSCCompressPictureFileSelect                = 0x000D
kSCRequestSequenceSettingsSelect            = 0x000E
kSCCompressSequenceBeginSelect              = 0x000F
kSCCompressSequenceFrameSelect              = 0x0010
kSCCompressSequenceEndSelect                = 0x0011
kSCDefaultPictHandleSettingsSelect          = 0x0012
kSCDefaultPictFileSettingsSelect            = 0x0013
kSCDefaultPixMapSettingsSelect              = 0x0014
kSCGetInfoSelect                            = 0x0015
kSCSetInfoSelect                            = 0x0016
kSCNewGWorldSelect                          = 0x0017
kSCSetCompressFlagsSelect                   = 0x0018
kSCGetCompressFlagsSelect                   = 0x0019
kSCGetSettingsAsTextSelect                  = 0x001A
kSCGetSettingsAsAtomContainerSelect         = 0x001B
kSCSetSettingsFromAtomContainerSelect       = 0x001C
kSCCompressSequenceFrameAsyncSelect         = 0x001D
kSCAsyncIdleSelect                          = 0x001E
kSCCopyCompressionSessionOptionsSelect      = 0x001F
kSCAudioInvokeLegacyCodecOptionsDialogSelect = 0x0081
kTweenerInitializeSelect                    = 0x0001
kTweenerDoTweenSelect                       = 0x0002
kTweenerResetSelect                         = 0x0003
kTCGetCurrentTimeCodeSelect                 = 0x0101
kTCGetTimeCodeAtTimeSelect                  = 0x0102
kTCTimeCodeToStringSelect                   = 0x0103
kTCTimeCodeToFrameNumberSelect              = 0x0104
kTCFrameNumberToTimeCodeSelect              = 0x0105
kTCGetSourceRefSelect                       = 0x0106
kTCSetSourceRefSelect                       = 0x0107
kTCSetTimeCodeFlagsSelect                   = 0x0108
kTCGetTimeCodeFlagsSelect                   = 0x0109
kTCSetDisplayOptionsSelect                  = 0x010A
kTCGetDisplayOptionsSelect                  = 0x010B
kMovieImportHandleSelect                    = 0x0001
kMovieImportFileSelect                      = 0x0002
```

```
kMovieImportSetSampleDurationSelect        = 0x0003
kMovieImportSetSampleDescriptionSelect     = 0x0004
kMovieImportSetMediaFileSelect             = 0x0005
kMovieImportSetDimensionsSelect            = 0x0006
kMovieImportSetChunkSizeSelect             = 0x0007
kMovieImportSetProgressProcSelect          = 0x0008
kMovieImportSetAuxiliaryDataSelect         = 0x0009
kMovieImportSetFromScrapSelect             = 0x000A
kMovieImportDoUserDialogSelect             = 0x000B
kMovieImportSetDurationSelect              = 0x000C
kMovieImportGetAuxiliaryDataTypeSelect     = 0x000D
kMovieImportValidateSelect                 = 0x000E
kMovieImportGetFileTypeSelect              = 0x000F
kMovieImportDataRefSelect                  = 0x0010
kMovieImportGetSampleDescriptionSelect     = 0x0011
kMovieImportGetMIMETypeListSelect          = 0x0012
kMovieImportSetOffsetAndLimitSelect        = 0x0013
kMovieImportGetSettingsAsAtomContainerSelect = 0x0014
kMovieImportSetSettingsFromAtomContainerSelect = 0x0015
kMovieImportSetOffsetAndLimit64Select      = 0x0016
kMovieImportIdleSelect                     = 0x0017
kMovieImportValidateDataRefSelect          = 0x0018
kMovieImportGetLoadStateSelect             = 0x0019
kMovieImportGetMaxLoadedTimeSelect         = 0x001A
kMovieImportEstimateCompletionTimeSelect   = 0x001B
kMovieImportSetDontBlockSelect             = 0x001C
kMovieImportGetDontBlockSelect             = 0x001D
kMovieImportSetIdleManagerSelect           = 0x001E
kMovieImportSetNewMovieFlagsSelect         = 0x001F
kMovieImportGetDestinationMediaTypeSelect  = 0x0020
kMovieImportSetMediaDataRefSelect          = 0x0021
kMovieImportDoUserDialogDataRefSelect      = 0x0022
kMovieExportToHandleSelect                 = 0x0080
kMovieExportToFileSelect                   = 0x0081
kMovieExportGetAuxiliaryDataSelect         = 0x0083
kMovieExportSetProgressProcSelect          = 0x0084
kMovieExportSetSampleDescriptionSelect     = 0x0085
kMovieExportDoUserDialogSelect             = 0x0086
kMovieExportGetCreatorTypeSelect           = 0x0087
kMovieExportToDataRefSelect                = 0x0088
kMovieExportFromProceduresToDataRefSelect  = 0x0089
kMovieExportAddDataSourceSelect            = 0x008A
kMovieExportValidateSelect                 = 0x008B
kMovieExportGetSettingsAsAtomContainerSelect = 0x008C
kMovieExportSetSettingsFromAtomContainerSelect = 0x008D
kMovieExportGetFileNameExtensionSelect     = 0x008E
kMovieExportGetShortFileTypeStringSelect   = 0x008F
kMovieExportGetSourceMediaTypeSelect       = 0x0090
kMovieExportSetGetMoviePropertyProcSelect  = 0x0091
kTextExportGetDisplayDataSelect            = 0x0100
kTextExportGetTimeFractionSelect           = 0x0101
kTextExportSetTimeFractionSelect           = 0x0102
kTextExportGetSettingsSelect               = 0x0103
kTextExportSetSettingsSelect               = 0x0104
kMIDIImportGetSettingsSelect               = 0x0100
kMIDIImportSetSettingsSelect               = 0x0101
kMovieExportNewGetDataAndPropertiesProcsSelect = 0x0100
kMovieExportDisposeGetDataAndPropertiesProcsSelect = 0x0101
```

```
kGraphicsImageImportSetSequenceEnabledSelect = 0x0100
kGraphicsImageImportGetSequenceEnabledSelect = 0x0101
kPreviewShowDataSelect                     = 0x0001
kPreviewMakePreviewSelect                  = 0x0002
kPreviewMakePreviewReferenceSelect         = 0x0003
kPreviewEventSelect                        = 0x0004
kDataCodecDecompressSelect                 = 0x0001
kDataCodecGetCompressBufferSizeSelect      = 0x0002
kDataCodecCompressSelect                   = 0x0003
kDataCodecBeginInterruptSafeSelect         = 0x0004
kDataCodecEndInterruptSafeSelect           = 0x0005
kDataCodecDecompressPartialSelect          = 0x0006
kDataCodecCompressPartialSelect            = 0x0007
kDataHGetDataSelect                        = 0x0002
kDataHPutDataSelect                        = 0x0003
kDataHFlushDataSelect                      = 0x0004
kDataHOpenForWriteSelect                   = 0x0005
kDataHCloseForWriteSelect                  = 0x0006
kDataHOpenForReadSelect                    = 0x0008
kDataHCloseForReadSelect                   = 0x0009
kDataHSetDataRefSelect                     = 0x000A
kDataHGetDataRefSelect                     = 0x000B
kDataHCompareDataRefSelect                 = 0x000C
kDataHTaskSelect                           = 0x000D
kDataHScheduleDataSelect                   = 0x000E
kDataHFinishDataSelect                     = 0x000F
kDataHFlushCacheSelect                     = 0x0010
kDataHResolveDataRefSelect                 = 0x0011
kDataHGetFileSizeSelect                    = 0x0012
kDataHCanUseDataRefSelect                  = 0x0013
kDataHGetVolumeListSelect                  = 0x0014
kDataHWriteSelect                          = 0x0015
kDataHPreextendSelect                      = 0x0016
kDataHSetFileSizeSelect                    = 0x0017
kDataHGetFreeSpaceSelect                   = 0x0018
kDataHCreateFileSelect                     = 0x0019
kDataHGetPreferredBlockSizeSelect          = 0x001A
kDataHGetDeviceIndexSelect                 = 0x001B
kDataHIsStreamingDataHandlerSelect         = 0x001C
kDataHGetDataInBufferSelect                = 0x001D
kDataHGetScheduleAheadTimeSelect           = 0x001E
kDataHSetCacheSizeLimitSelect              = 0x001F
kDataHGetCacheSizeLimitSelect              = 0x0020
kDataHGetMovieSelect                       = 0x0021
kDataHAddMovieSelect                       = 0x0022
kDataHUpdateMovieSelect                    = 0x0023
kDataHDoesBufferSelect                     = 0x0024
kDataHGetFileNameSelect                    = 0x0025
kDataHGetAvailableFileSizeSelect           = 0x0026
kDataHGetMacOSFileTypeSelect               = 0x0027
kDataHGetMIMETypeSelect                    = 0x0028
kDataHSetDataRefWithAnchorSelect           = 0x0029
kDataHGetDataRefWithAnchorSelect           = 0x002A
kDataHSetMacOSFileTypeSelect               = 0x002B
kDataHSetTimeBaseSelect                    = 0x002C
kDataHGetInfoFlagsSelect                   = 0x002D
kDataHScheduleData64Select                 = 0x002E
kDataHWrite64Select                        = 0x002F
```

```
kDataHGetFileSize64Select                 = 0x0030
kDataHPreextend64Select                   = 0x0031
kDataHSetFileSize64Select                 = 0x0032
kDataHGetFreeSpace64Select                = 0x0033
kDataHAppend64Select                      = 0x0034
kDataHReadAsyncSelect                     = 0x0035
kDataHPollReadSelect                      = 0x0036
kDataHGetDataAvailabilitySelect           = 0x0037
kDataHGetFileSizeAsyncSelect              = 0x003A
kDataHGetDataRefAsTypeSelect              = 0x003B
kDataHSetDataRefExtensionSelect           = 0x003C
kDataHGetDataRefExtensionSelect           = 0x003D
kDataHGetMovieWithFlagsSelect             = 0x003E
kDataHGetFileTypeOrderingSelect           = 0x0040
kDataHCreateFileWithFlagsSelect           = 0x0041
kDataHGetMIMETypeAsyncSelect              = 0x0042
kDataHGetInfoSelect                       = 0x0043
kDataHSetIdleManagerSelect                = 0x0044
kDataHDeleteFileSelect                    = 0x0045
kDataHSetMovieUsageFlagsSelect            = 0x0046
kDataHUseTemporaryDataRefSelect           = 0x0047
kDataHGetTemporaryDataRefCapabilitiesSelect = 0x0048
kDataHRenameFileSelect                    = 0x0049
kDataHGetAvailableFileSize64Select        = 0x004E
kDataHGetDataAvailability64Select         = 0x004F
kDataHPlaybackHintsSelect                 = 0x0103
kDataHPlaybackHints64Select               = 0x010E
kDataHGetDataRateSelect                   = 0x0110
kDataHSetTimeHintsSelect                  = 0x0111
kVDGetMaxSrcRectSelect                    = 0x0001
kVDGetActiveSrcRectSelect                 = 0x0002
kVDSetDigitizerRectSelect                 = 0x0003
kVDGetDigitizerRectSelect                 = 0x0004
kVDGetVBlankRectSelect                    = 0x0005
kVDGetMaskPixMapSelect                    = 0x0006
kVDGetPlayThruDestinationSelect           = 0x0008
kVDUseThisCLUTSelect                      = 0x0009
kVDSetInputGammaValueSelect               = 0x000A
kVDGetInputGammaValueSelect               = 0x000B
kVDSetBrightnessSelect                    = 0x000C
kVDGetBrightnessSelect                    = 0x000D
kVDSetContrastSelect                      = 0x000E
kVDSetHueSelect                           = 0x000F
kVDSetSharpnessSelect                     = 0x0010
kVDSetSaturationSelect                    = 0x0011
kVDGetContrastSelect                      = 0x0012
kVDGetHueSelect                           = 0x0013
kVDGetSharpnessSelect                     = 0x0014
kVDGetSaturationSelect                    = 0x0015
kVDGrabOneFrameSelect                     = 0x0016
kVDGetMaxAuxBufferSelect                  = 0x0017
kVDGetDigitizerInfoSelect                 = 0x0019
kVDGetCurrentFlagsSelect                  = 0x001A
kVDSetKeyColorSelect                      = 0x001B
kVDGetKeyColorSelect                      = 0x001C
kVDAddKeyColorSelect                      = 0x001D
kVDGetNextKeyColorSelect                  = 0x001E
kVDSetKeyColorRangeSelect                 = 0x001F
```

```
kVDGetKeyColorRangeSelect                   = 0x0020
kVDSetDigitizerUserInterruptSelect          = 0x0021
kVDSetInputColorSpaceModeSelect             = 0x0022
kVDGetInputColorSpaceModeSelect             = 0x0023
kVDSetClipStateSelect                       = 0x0024
kVDGetClipStateSelect                       = 0x0025
kVDSetClipRgnSelect                         = 0x0026
kVDClearClipRgnSelect                       = 0x0027
kVDGetCLUTInUseSelect                       = 0x0028
kVDSetPLLFilterTypeSelect                   = 0x0029
kVDGetPLLFilterTypeSelect                   = 0x002A
kVDGetMaskandValueSelect                    = 0x002B
kVDSetMasterBlendLevelSelect                = 0x002C
kVDSetPlayThruDestinationSelect             = 0x002D
kVDSetPlayThruOnOffSelect                   = 0x002E
kVDSetFieldPreferenceSelect                 = 0x002F
kVDGetFieldPreferenceSelect                 = 0x0030
kVDPreflightDestinationSelect               = 0x0032
kVDPreflightGlobalRectSelect                = 0x0033
kVDSetPlayThruGlobalRectSelect              = 0x0034
kVDSetInputGammaRecordSelect                = 0x0035
kVDGetInputGammaRecordSelect                = 0x0036
kVDSetBlackLevelValueSelect                 = 0x0037
kVDGetBlackLevelValueSelect                 = 0x0038
kVDSetWhiteLevelValueSelect                 = 0x0039
kVDGetWhiteLevelValueSelect                 = 0x003A
kVDGetVideoDefaultsSelect                   = 0x003B
kVDGetNumberOfInputsSelect                  = 0x003C
kVDGetInputFormatSelect                     = 0x003D
kVDSetInputSelect                           = 0x003E
kVDGetInputSelect                           = 0x003F
kVDSetInputStandardSelect                   = 0x0040
kVDSetupBuffersSelect                       = 0x0041
kVDGrabOneFrameAsyncSelect                  = 0x0042
kVDDoneSelect                               = 0x0043
kVDSetCompressionSelect                     = 0x0044
kVDCompressOneFrameAsyncSelect              = 0x0045
kVDCompressDoneSelect                       = 0x0046
kVDReleaseCompressBufferSelect              = 0x0047
kVDGetImageDescriptionSelect                = 0x0048
kVDResetCompressSequenceSelect              = 0x0049
kVDSetCompressionOnOffSelect                = 0x004A
kVDGetCompressionTypesSelect                = 0x004B
kVDSetTimeBaseSelect                        = 0x004C
kVDSetFrameRateSelect                       = 0x004D
kVDGetDataRateSelect                        = 0x004E
kVDGetSoundInputDriverSelect                = 0x004F
kVDGetDMADepthsSelect                       = 0x0050
kVDGetPreferredTimeScaleSelect              = 0x0051
kVDReleaseAsyncBuffersSelect                = 0x0052
kVDSetDataRateSelect                        = 0x0054
kVDGetTimeCodeSelect                        = 0x0055
kVDUseSafeBuffersSelect                     = 0x0056
kVDGetSoundInputSourceSelect                = 0x0057
kVDGetCompressionTimeSelect                 = 0x0058
kVDSetPreferredPacketSizeSelect             = 0x0059
kVDSetPreferredImageDimensionsSelect        = 0x005A
kVDGetPreferredImageDimensionsSelect        = 0x005B
```

```
kVDGetInputNameSelect                      = 0x005C
kVDSetDestinationPortSelect                = 0x005D
kVDGetDeviceNameAndFlagsSelect             = 0x005E
kVDCaptureStateChangingSelect              = 0x005F
kVDGetUniqueIDsSelect                      = 0x0060
kVDSelectUniqueIDsSelect                   = 0x0061
kVDCopyPreferredAudioDeviceSelect          = 0x0063
kVDIIDCGetFeaturesSelect                   = 0x0200
kVDIIDCSetFeaturesSelect                   = 0x0201
kVDIIDCGetDefaultFeaturesSelect            = 0x0202
kVDIIDCGetCSRDataSelect                    = 0x0203
kVDIIDCSetCSRDataSelect                    = 0x0204
kVDIIDCGetFeaturesForSpecifierSelect       = 0x0205
kXMLParseDataRefSelect                     = 0x0001
kXMLParseFileSelect                        = 0x0002
kXMLParseDisposeXMLDocSelect               = 0x0003
kXMLParseGetDetailedParseErrorSelect       = 0x0004
kXMLParseAddElementSelect                  = 0x0005
kXMLParseAddAttributeSelect                = 0x0006
kXMLParseAddMultipleAttributesSelect       = 0x0007
kXMLParseAddAttributeAndValueSelect        = 0x0008
kXMLParseAddMultipleAttributesAndValuesSelect = 0x0009
kXMLParseAddAttributeValueKindSelect       = 0x000A
kXMLParseAddNameSpaceSelect                = 0x000B
kXMLParseSetOffsetAndLimitSelect           = 0x000C
kXMLParseSetEventParseRefConSelect         = 0x000D
kXMLParseSetStartDocumentHandlerSelect     = 0x000E
kXMLParseSetEndDocumentHandlerSelect       = 0x000F
kXMLParseSetStartElementHandlerSelect      = 0x0010
kXMLParseSetEndElementHandlerSelect        = 0x0011
kXMLParseSetCharDataHandlerSelect          = 0x0012
kXMLParseSetPreprocessInstructionHandlerSelect = 0x0013
kXMLParseSetCommentHandlerSelect           = 0x0014
kXMLParseSetCDataHandlerSelect             = 0x0015
kSGInitializeSelect                        = 0x0001
kSGSetDataOutputSelect                     = 0x0002
kSGGetDataOutputSelect                     = 0x0003
kSGSetGWorldSelect                         = 0x0004
kSGGetGWorldSelect                         = 0x0005
kSGNewChannelSelect                        = 0x0006
kSGDisposeChannelSelect                    = 0x0007
kSGStartPreviewSelect                      = 0x0010
kSGStartRecordSelect                       = 0x0011
kSGIdleSelect                              = 0x0012
kSGStopSelect                              = 0x0013
kSGPauseSelect                             = 0x0014
kSGPrepareSelect                           = 0x0015
kSGReleaseSelect                           = 0x0016
kSGGetMovieSelect                          = 0x0017
kSGSetMaximumRecordTimeSelect              = 0x0018
kSGGetMaximumRecordTimeSelect              = 0x0019
kSGGetStorageSpaceRemainingSelect          = 0x001A
kSGGetTimeRemainingSelect                  = 0x001B
kSGGrabPictSelect                          = 0x001C
kSGGetLastMovieResIDSelect                 = 0x001D
kSGSetFlagsSelect                          = 0x001E
kSGGetFlagsSelect                          = 0x001F
kSGSetDataProcSelect                       = 0x0020
```

```
kSGNewChannelFromComponentSelect           = 0x0021
kSGDisposeDeviceListSelect                 = 0x0022
kSGAppendDeviceListToMenuSelect            = 0x0023
kSGSetSettingsSelect                       = 0x0024
kSGGetSettingsSelect                       = 0x0025
kSGGetIndChannelSelect                     = 0x0026
kSGUpdateSelect                            = 0x0027
kSGGetPauseSelect                          = 0x0028
kSGSettingsDialogSelect                    = 0x0029
kSGGetAlignmentProcSelect                  = 0x002A
kSGSetChannelSettingsSelect                = 0x002B
kSGGetChannelSettingsSelect                = 0x002C
kSGGetModeSelect                           = 0x002D
kSGSetDataRefSelect                        = 0x002E
kSGGetDataRefSelect                        = 0x002F
kSGNewOutputSelect                         = 0x0030
kSGDisposeOutputSelect                     = 0x0031
kSGSetOutputFlagsSelect                    = 0x0032
kSGSetChannelOutputSelect                  = 0x0033
kSGGetDataOutputStorageSpaceRemainingSelect = 0x0034
kSGHandleUpdateEventSelect                 = 0x0035
kSGSetOutputNextOutputSelect               = 0x0036
kSGGetOutputNextOutputSelect               = 0x0037
kSGSetOutputMaximumOffsetSelect            = 0x0038
kSGGetOutputMaximumOffsetSelect            = 0x0039
kSGGetOutputDataReferenceSelect            = 0x003A
kSGWriteExtendedMovieDataSelect            = 0x003B
kSGGetStorageSpaceRemaining64Select        = 0x003C
kSGGetDataOutputStorageSpaceRemaining64Select = 0x003D
kSGWriteMovieDataSelect                    = 0x0100
kSGAddFrameReferenceSelect                 = 0x0101
kSGGetNextFrameReferenceSelect             = 0x0102
kSGGetTimeBaseSelect                       = 0x0103
kSGSortDeviceListSelect                    = 0x0104
kSGAddMovieDataSelect                      = 0x0105
kSGChangedSourceSelect                     = 0x0106
kSGAddExtendedFrameReferenceSelect         = 0x0107
kSGGetNextExtendedFrameReferenceSelect     = 0x0108
kSGAddExtendedMovieDataSelect              = 0x0109
kSGAddOutputDataRefToMediaSelect           = 0x010A
kSGSetSettingsSummarySelect                = 0x010B
kSGSetChannelUsageSelect                   = 0x0080
kSGGetChannelUsageSelect                   = 0x0081
kSGSetChannelBoundsSelect                  = 0x0082
kSGGetChannelBoundsSelect                  = 0x0083
kSGSetChannelVolumeSelect                  = 0x0084
kSGGetChannelVolumeSelect                  = 0x0085
kSGGetChannelInfoSelect                    = 0x0086
kSGSetChannelPlayFlagsSelect               = 0x0087
kSGGetChannelPlayFlagsSelect               = 0x0088
kSGSetChannelMaxFramesSelect               = 0x0089
kSGGetChannelMaxFramesSelect               = 0x008A
kSGSetChannelRefConSelect                  = 0x008B
kSGSetChannelClipSelect                    = 0x008C
kSGGetChannelClipSelect                    = 0x008D
kSGGetChannelSampleDescriptionSelect       = 0x008E
kSGGetChannelDeviceListSelect              = 0x008F
kSGSetChannelDeviceSelect                  = 0x0090
```

```
kSGSetChannelMatrixSelect                  = 0x0091
kSGGetChannelMatrixSelect                  = 0x0092
kSGGetChannelTimeScaleSelect               = 0x0093
kSGChannelPutPictureSelect                 = 0x0094
kSGChannelSetRequestedDataRateSelect       = 0x0095
kSGChannelGetRequestedDataRateSelect       = 0x0096
kSGChannelSetDataSourceNameSelect          = 0x0097
kSGChannelGetDataSourceNameSelect          = 0x0098
kSGChannelSetCodecSettingsSelect           = 0x0099
kSGChannelGetCodecSettingsSelect           = 0x009A
kSGGetChannelTimeBaseSelect                = 0x009B
kSGGetChannelRefConSelect                  = 0x009C
kSGGetChannelDeviceAndInputNamesSelect     = 0x009D
kSGSetChannelDeviceInputSelect             = 0x009E
kSGSetChannelSettingsStateChangingSelect   = 0x009F
kSGInitChannelSelect                       = 0x0180
kSGWriteSamplesSelect                      = 0x0181
kSGGetDataRateSelect                       = 0x0182
kSGAlignChannelRectSelect                  = 0x0183
kSGPanelGetDitlSelect                      = 0x0200
kSGPanelGetTitleSelect                     = 0x0201
kSGPanelCanRunSelect                       = 0x0202
kSGPanelInstallSelect                      = 0x0203
kSGPanelEventSelect                        = 0x0204
kSGPanelItemSelect                         = 0x0205
kSGPanelRemoveSelect                       = 0x0206
kSGPanelSetGrabberSelect                   = 0x0207
kSGPanelSetResFileSelect                   = 0x0208
kSGPanelGetSettingsSelect                  = 0x0209
kSGPanelSetSettingsSelect                  = 0x020A
kSGPanelValidateInputSelect                = 0x020B
kSGPanelSetEventFilterSelect               = 0x020C
kSGPanelGetDITLForSizeSelect               = 0x020D
kSGGetSrcVideoBoundsSelect                 = 0x0100
kSGSetVideoRectSelect                      = 0x0101
kSGGetVideoRectSelect                      = 0x0102
kSGGetVideoCompressorTypeSelect            = 0x0103
kSGSetVideoCompressorTypeSelect            = 0x0104
kSGSetVideoCompressorSelect                = 0x0105
kSGGetVideoCompressorSelect                = 0x0106
kSGGetVideoDigitizerComponentSelect        = 0x0107
kSGSetVideoDigitizerComponentSelect        = 0x0108
kSGVideoDigitizerChangedSelect             = 0x0109
kSGSetVideoBottlenecksSelect               = 0x010A
kSGGetVideoBottlenecksSelect               = 0x010B
kSGGrabFrameSelect                         = 0x010C
kSGGrabFrameCompleteSelect                 = 0x010D
kSGDisplayFrameSelect                      = 0x010E
kSGCompressFrameSelect                     = 0x010F
kSGCompressFrameCompleteSelect             = 0x0110
kSGAddFrameSelect                          = 0x0111
kSGTransferFrameForCompressSelect          = 0x0112
kSGSetCompressBufferSelect                 = 0x0113
kSGGetCompressBufferSelect                 = 0x0114
kSGGetBufferInfoSelect                     = 0x0115
kSGSetUseScreenBufferSelect                = 0x0116
kSGGetUseScreenBufferSelect                = 0x0117
kSGGrabCompressCompleteSelect              = 0x0118
```

```
kSGDisplayCompressSelect                    = 0x0119
kSGSetFrameRateSelect                       = 0x011A
kSGGetFrameRateSelect                       = 0x011B
kSGSetPreferredPacketSizeSelect             = 0x0121
kSGGetPreferredPacketSizeSelect             = 0x0122
kSGSetUserVideoCompressorListSelect         = 0x0123
kSGGetUserVideoCompressorListSelect         = 0x0124
kSGSetSoundInputDriverSelect                = 0x0100
kSGGetSoundInputDriverSelect                = 0x0101
kSGSoundInputDriverChangedSelect            = 0x0102
kSGSetSoundRecordChunkSizeSelect            = 0x0103
kSGGetSoundRecordChunkSizeSelect            = 0x0104
kSGSetSoundInputRateSelect                  = 0x0105
kSGGetSoundInputRateSelect                  = 0x0106
kSGSetSoundInputParametersSelect            = 0x0107
kSGGetSoundInputParametersSelect            = 0x0108
kSGSetAdditionalSoundRatesSelect            = 0x0109
kSGGetAdditionalSoundRatesSelect            = 0x010A
kSGSetFontNameSelect                        = 0x0100
kSGSetFontSizeSelect                        = 0x0101
kSGSetTextForeColorSelect                   = 0x0102
kSGSetTextBackColorSelect                   = 0x0103
kSGSetJustificationSelect                   = 0x0104
kSGGetTextReturnToSpaceValueSelect          = 0x0105
kSGSetTextReturnToSpaceValueSelect          = 0x0106
kSGGetInstrumentSelect                      = 0x0100
kSGSetInstrumentSelect                      = 0x0101
kQTVideoOutputGetDisplayModeListSelect      = 0x0001
kQTVideoOutputGetCurrentClientNameSelect    = 0x0002
kQTVideoOutputSetClientNameSelect           = 0x0003
kQTVideoOutputGetClientNameSelect           = 0x0004
kQTVideoOutputBeginSelect                   = 0x0005
kQTVideoOutputEndSelect                     = 0x0006
kQTVideoOutputSetDisplayModeSelect          = 0x0007
kQTVideoOutputGetDisplayModeSelect          = 0x0008
kQTVideoOutputSaveStateSelect               = 0x000A
kQTVideoOutputRestoreStateSelect            = 0x000B
kQTVideoOutputGetGWorldSelect               = 0x000C
kQTVideoOutputGetGWorldParametersSelect     = 0x000D
kQTVideoOutputGetIndSoundOutputSelect       = 0x000E
kQTVideoOutputGetClockSelect                = 0x000F
kQTVideoOutputSetEchoPortSelect             = 0x0010
kQTVideoOutputGetIndImageDecompressorSelect = 0x0011
kQTVideoOutputBaseSetEchoPortSelect         = 0x0012
kQTVideoOutputCopyIndAudioOutputDeviceUIDSelect = 0x0016
```

Component Identifiers

Identify the types of components.

```
clockComponentType                =' clok'
compressorComponentType           ='imco'
CreateFilePreviewComponentType    ='pmak'
DataHandlerType                   ='dhlr'
decompressorComponentType         ='imdc'
MediaHandlerType                  ='mhlr'
MovieControllerComponentType      ='play'
MovieExportType                   ='spit'
MovieImportType                   ='eat '
SeqGrabChannelType                ='sgch'
SeqGrabComponentType              ='barg'
SeqGrabCompressionPanelType       ='cmpr'
SeqGrabPanelType                  ='sgpn'
SeqGrabSourcePanelType            ='sour'
ShowFilePreviewComponentType      ='pnot'
StandardCompressionSubType        ='imag'
StandardCompressionSubTypeSound   ='soun'
StandardCompressionType           ='scdi'
systemMicrosecondClock            ='micr'
systemMillisecondClock            ='mill'
systemSecondClock                 ='seco'
systemTickClock                   ='tick'
videoDigitizerComponentType       ='vdig'
```

Discussion
All components of the same type or subtype provide the same kinds of services and support a common application programming interface. Codecs have their own set of types.

Component Property IDs and Flags

Constants that contain the flags and IDs of component properties.

```
uppCallComponentGetComponentPropertyInfoProcInfo = 0x0003FFF0
uppCallComponentGetComponentPropertyProcInfo = 0x0003FFF0
uppCallComponentSetComponentPropertyProcInfo = 0x0000FFF0
uppCallComponentAddComponentPropertyListenerProcInfo = 0x0000FFF0
uppCallComponentRemoveComponentPropertyListenerProcInfo = 0x0000FFF0
kCallComponentExecuteWiredActionSelect     = -9
kComponentPropertyFlagCanSetLater = (1L << 0)
kComponentPropertyFlagCanSetNow = (1L << 1)
kComponentPropertyFlagCanGetNow = (1L << 3)
kComponentPropertyFlagHasExtendedInfo = (1L << 4)
kComponentPropertyFlagValueMustBeReleased = (1L << 5)
kComponentPropertyFlagValueIsCFTypeRef = (1L << 6)
kComponentPropertyFlagGetBufferMustBeInitialized = (1L << 7)
kQTComponentPropertyListenerCollectionContextVersion = 1
kQTGetComponentPropertyInfoSelect          = -11
kQTGetComponentPropertySelect              = -12
kQTSetComponentPropertySelect              = -13
kQTAddComponentPropertyListenerSelect      = -14
kQTRemoveComponentPropertyListenerSelect   = -15
```

Error Codes

Identify errors generated while executing QuickTime calls.

```
// General QuickTime errors
couldNotResolveDataRef                  =-2000
badImageDescription                     =-2001
badPublicMovieAtom                      =-2002
cantFindHandler                         =-2003
cantOpenHandler                         =-2004
badComponentType                        =-2005
noMediaHandler                          =-2006
noDataHandler                           =-2007
invalidMedia                            =-2008
invalidTrack                            =-2009
invalidMovie                            =-2010
invalidSampleTable                      =-2011
invalidDataRef                          =-2012
invalidHandler                          =-2013
invalidDuration                         =-2014
invalidTime                             =-2015
cantPutPublicMovieAtom                  =-2016
badEditList                             =-2017
mediaTypesDontMatch                     =-2018
progressProcAborted                     =-2019
movieToolboxUninitialized               =-2020
noRecordOfApp                           =-2020
wfFileNotFound                          =-2021
cantCreateSingleForkFile                =-2022
invalidEditState                        =-2023
nonMatchingEditState                    =-2024
staleEditState                          =-2025
userDataItemNotFound                    =-2026
maxSizeToGrowTooSmall                   =-2027
badTrackIndex                           =-2028
trackIDNotFound                         =-2029
trackNotInMovie                         =-2030
timeNotInTrack                          =-2031
timeNotInMedia                          =-2032
badEditIndex                            =-2033
internalQuickTimeError                  =-2034
cantEnableTrack                         =-2035
invalidRect                             =-2036
invalidSampleNum                        =-2037
invalidChunkNum                         =-2038
invalidSampleDescIndex                  =-2039
invalidChunkCache                       =-2040
invalidSampleDescription                =-2041
dataNotOpenForRead                      =-2042
dataNotOpenForWrite                     =-2043
dataAlreadyOpenForWrite                 =-2044
dataAlreadyClosed                       =-2045
endOfDataReached                        =-2046
dataNoDataRef                           =-2047
noMovieFound                            =-2048
invalidDataRefContainer                 =-2049
badDataRefIndex                         =-2050
noDefaultDataRef                        =-2051
couldNotUseAnExistingSample             =-2052
featureUnsupported                      =-2053
unsupportedAuxiliaryImportData          =-2057
auxiliaryExportDataUnavailable          =-2058
```

```
samplesAlreadyInMediaErr            =-2059
noSourceTreeFoundErr                =-2060
sourceNotFoundErr                   =-2061
movieTextNotFoundErr                =-2062
missingRequiredParameterErr         =-2063
invalidSpriteWorldPropertyErr       =-2064
invalidSpritePropertyErr            =-2065
gWorldsNotSameDepthAndSizeErr       =-2066
invalidSpriteIndexErr               =-2067
invalidImageIndexErr                =-2068
invalidSpriteIDErr                  =-2069
// QuickTime Music Architecture errors
internalComponentErr                =-2070
notImplementedMusicOSErr            =-2071
cantSendToSynthesizerOSErr          =-2072
cantReceiveFromSynthesizerOSErr     =-2073
illegalVoiceAllocationOSErr         =-2074
illegalPartOSErr                    =-2075
illegalChannelOSErr                 =-2076
illegalKnobOSErr                    =-2077
illegalKnobValueOSErr               =-2078
illegalInstrumentOSErr              =-2079
illegalControllerOSErr              =-2080
midiManagerAbsentOSErr              =-2081
synthesizerNotRespondingOSErr       =-2082
synthesizerOSErr                    =-2083
illegalNoteChannelOSErr             =-2084
noteChannelNotAllocatedOSErr        =-2085
tunePlayerFullOSErr                 =-2086
tuneParseOSErr                      =-2087
noExportProcAvailableErr            =-2089
videoOutputInUseErr                 =-2090
// Windows-specific errors
componentDllLoadErr                 =-2091
componentDllEntryNotFoundErr        =-2092
qtmlDllLoadErr                      =-2093
qtmlDllEntryNotFoundErr             =-2094
qtmlUninitialized                   =-2095
unsupportedOSErr                    =-2096
unsupportedProcessorErr             =-2097
noVideoTrackInMovieErr              =-2054
noSoundTrackInMovieErr              =-2055
soundSupportNotAvailableErr         =-2056
// QT atom errors
cannotFindAtomErr                   =-2101
notLeafAtomErr                      =-2102
atomsNotOfSameTypeErr               =-2103
atomIndexInvalidErr                 =-2104
duplicateAtomTypeAndIDErr           =-2105
invalidAtomErr                      =-2106
invalidAtomContainerErr             =-2107
invalidAtomTypeErr                  =-2108
cannotBeLeafAtomErr                 =-2109
// Data access errors
pathTooLongErr                      =-2110
emptyPathErr                        =-2111
noPathMappingErr                    =-2112
pathNotVerifiedErr                  =-2113
```

```
unknownFormatErr                    =-2114
wackBadFileErr                      =-2115
wackForkNotFoundErr                 =-2116
wackBadMetaDataErr                  =-2117
qfcbNotFoundErr                     =-2118
qfcbNotCreatedErr                   =-2119
AAPNotCreatedErr                    =-2120
AAPNotFoundErr                      =-2121
ASDBadHeaderErr                     =-2122
ASDBadForkErr                       =-2123
ASDEntryNotFoundErr                 =-2124
fileOffsetTooBigErr                 =-2125
notAllowedToSaveMovieErr            =-2126
qtNetworkAlreadyAllocatedErr        =-2127
urlDataHHTTPProtocolErr             =-2129
urlDataHHTTPNoNetDriverErr          =-2130
urlDataHHTTPURLErr                  =-2131
urlDataHHTTPRedirectErr             =-2132
urlDataHFTPProtocolErr              =-2133
urlDataHFTPShutdownErr              =-2134
urlDataHFTPBadUserErr               =-2135
urlDataHFTPBadPasswordErr           =-2136
urlDataHFTPServerErr                =-2137
urlDataHFTPDataConnectionErr        =-2138
urlDataHFTPNoDirectoryErr           =-2139
urlDataHFTPQuotaErr                 =-2140
urlDataHFTPPermissionsErr           =-2141
urlDataHFTPFilenameErr              =-2142
urlDataHFTPNoNetDriverErr           =-2143
urlDataHFTPBadNameListErr           =-2144
urlDataHFTPNeedPasswordErr          =-2145
urlDataHFTPNoPasswordErr            =-2146
urlDataHFTPServerDisconnectedErr    =-2147
urlDataHFTPURLErr                   =-2148
notEnoughDataErr                    =-2149
qtActionNotHandledErr               =-2157
// Digitizing errors
digiUnimpErr                        =-2201
qtParamErr                          =-2202
matrixErr                           =-2203
notExactMatrixErr                   =-2204
noMoreKeyColorsErr                  =-2205
notExactSizeErr                     =-2206
badDepthErr                         =-2207
noDMAErr                            =-2208
badCallOrderErr                     =-2209
// Codec errors
codecErr                            =-8960
noCodecErr                          =-8961
codecUnimpErr                       =-8962
codecSizeErr                        =-8963
codecScreenBufErr                   =-8964
codecImageBufErr                    =-8965
codecSpoolErr                       =-8966
codecAbortErr                       =-8967
codecWouldOffscreenErr              =-8968
codecBadDataErr                     =-8969
codecDataVersErr                    =-8970
```

```
codecExtensionNotFoundErr             =-8971
scTypeNotFoundErr                     =-8971
codecConditionErr                     =-8972
codecOpenErr                          =-8973
codecCantWhenErr                      =-8974
codecCantQueueErr                     =-8975
codecNothingToBlitErr                 =-8976
codecNoMemoryPleaseWaitErr            =-8977
codecDisabledErr                      =-8978
codecNeedToFlushChainErr              =-8979
lockPortBitsBadSurfaceErr             =-8980
lockPortBitsWindowMovedErr            =-8981
lockPortBitsWindowResizedErr          =-8982
lockPortBitsWindowClippedErr          =-8983
lockPortBitsBadPortErr                =-8984
lockPortBitsSurfaceLostErr            =-8985
codecParameterDialogConfirm           =-8986
codecNeedAccessKeyErr                 =-8987
codecOffscreenFailedErr               =-8988
codecDroppedFrameErr                  =-8989
directXObjectAlreadyExists            =-8990
lockPortBitsWrongGDeviceErr           =-8991
codecOffscreenFailedPleaseRetryErr    =-8992
// Sequence Grabber errors
noDeviceForChannel                    =-9400,
grabTimeComplete                      =-9401,
cantDoThatInCurrentMode               =-9402,
notEnoughMemoryToGrab                 =-9403,
notEnoughDiskSpaceToGrab              =-9404,
couldntGetRequiredComponent           =-9405,
badSGChannel                          =-9406,
seqGrabInfoNotAvailable               =-9407,
deviceCantMeetRequest                 =-9408,
// Movie Controller errors
badControllerHeight                   =-9994,
editingNotAllowed                     =-9995,
controllerBoundsNotExact              =-9996,
cannotSetWidthOfAttachedController    =-9997,
controllerHasFixedHeight              =-9998,
cannotMoveAttachedController          =-9999
// QuickTime VR Errors
notAQTVRMovieErr                      =-30540
constraintReachedErr                  =-30541
callNotSupportedByNodeErr             =-30542
selectorNotSupportedByNodeErr         =-30543
invalidNodeIDErr                      =-30544
invalidViewStateErr                   =-30545
timeNotInViewErr                      =-30546
propertyNotSupportedByNodeErr         =-30547
settingNotSupportedByNodeErr          =-30548
limitReachedErr                       =-30549
invalidNodeFormatErr                  =-30550
invalidHotSpotIDErr                   =-30551
noMemoryNodeFailedInitialize          =-30552
streamingNodeNotReadyErr              =-30553
qtvrLibraryLoadErr                    =-30554
qtvrUninitialized                     =-30555
```

`noRecordOfApp`
A replica of the movieToolboxUninitialized error.

Available in Mac OS X v10.0 and later.

Declared in `MacErrors.h`.

`cantCreateSingleForkFile`
The file to be created already exists.

Available in Mac OS X v10.0 and later.

Declared in `MacErrors.h`.

`componentDllLoadErr`
Windows error returned when a component is loading.

Available in Mac OS X v10.0 and later.

Declared in `MacErrors.h`.

`componentDllEntryNotFoundErr`
Windows error returned when a component is loading.

Available in Mac OS X v10.0 and later.

Declared in `MacErrors.h`.

`qtmlDllLoadErr`
Windows error returned when the QuickTime Media Layer is loading.

Available in Mac OS X v10.0 and later.

Declared in `MacErrors.h`.

`qtmlDllEntryNotFoundErr`
Windows error returned when the QuickTime Media Layer is loading.

Available in Mac OS X v10.0 and later.

Declared in `MacErrors.h`.

`digiUnimpErr`
Digitizer feature is unimplemented.

Available in Mac OS X v10.0 and later.

Declared in `MacErrors.h`.

`qtParamErr`
Bad input parameter (out of range, for example).

Available in Mac OS X v10.0 and later.

Declared in `MacErrors.h`.

`matrixErr`
Bad matrix; the digitizer did nothing.

Available in Mac OS X v10.0 and later.

Declared in `MacErrors.h`.

`notExactMatrixErr`
Warning of a bad matrix; the digitizer did its best.

Available in Mac OS X v10.0 and later.

Declared in `MacErrors.h`.

`noMoreKeyColorsErr`
All the key indexes are in use.

Available in Mac OS X v10.0 and later.

Declared in `MacErrors.h`.

`notExactSizeErr`
Can't digitize to the exact size requested.

Available in Mac OS X v10.0 and later.

Declared in `MacErrors.h`.

`badDepthErr`
Can't digitize into the requested pixel depth.

Available in Mac OS X v10.0 and later.

Declared in `MacErrors.h`.

`noDMAErr`
Can't do DMA digitizing; that is, can't go to the requested destination.

Available in Mac OS X v10.0 and later.

Declared in `MacErrors.h`.

`badCallOrderErr`
A status call was made before being set up first.

Available in Mac OS X v10.0 and later.

Declared in `MacErrors.h`.

Discussion
The Movie Toolbox provides two error values to your application: the current error and the sticky error. The current error is the result code from the last Movie Toolbox function; it is updated each time your application calls a Movie Toolbox function. The sticky error value contains the first nonzero result code from any Movie Toolbox function that you called after having cleared the sticky error with `ClearMoviesStickyError`.

File Types and Creators

Identify the formats of graphics files and the applications that create them.

```
// File types
ftAdobePremiereMovie       ='MooV'
ftAfterDarkModule          ='ADgm'
ftClip3Dgraphic            ='EZ3D'
ftCricketChart             ='CGPC'
ftCricketDrawing           ='CKDT'
ftDesignCADDrawing         ='DCAD'
ftImageStudioGraphic       ='RIFF'
ftKaleidaGraphGraphic      ='QPCT'
ftMacFlowChart             ='FLCH'
ftMacSpinDataSet           ='D2BN'
ftMoviePlayerMovie         ='MooV'
ftPixelPaint               ='PX01'
ftSuper3DDrawing           ='3DBX'
ftSwivel3DDrawing          ='SMDL'
ftVersaCADDrawing          ='2D  '
// Creator codes
sigAdobePremiere           ='PrMr'
sigAfterDark               ='ADrk'
sigAldusSuper3D            ='SP3D'
sigAutoCAD                 ='ACAD'
sigClip3D                  ='EZ3E'
sigCricketDraw             ='CRDW'
sigCricketGraph            ='CGRF'
sigDeltagraphPro           ='DGRH'
sigDesign2                 ='DESG'
sigDesignCAD               ='ASBC'
sigDesignStudio            ='MRJN'
sigDigDarkroom             ='DIDR'
sigDreams                  ='PHNX'
sigDynaperspective         ='PERS'
sigGenericCADD             ='CAD3'
sigGraphMaster             ='GRAM'
sigImageStudio             ='FSPE'
sigInfiniD                 ='SI∞D'
sigKaleidaGraph            ='QKPT'
sigKidPix                  ='Kid2'
sigLabVIEW                 ='LBVW'
sigMacDraft                ='MD20'
sigMacDraw                 ='MDRW'
sigMacFlow                 ='MCFL'
sigMacSpin                 ='D2SP'
sigMiniCad                 ='CDP3'
sigModelShop               ='MDSP'
sigMoviePlayer             ='TVOD'
sigMovieRecorder           ='mrcr'
sigOasis                   ='TAOA'
sigOBJECTMASTER            ='BROW'
sigOfoto                   ='APLS'
sigOmnis5                  ='Q2$$'
sigOptix                   ='PIXL'
sigPhotoMac                ='PMAC'
sigPictureCompressor       ='ppxi'
sigPICTViewer              ='MDTS'
sigPixelPaint              ='PIXR'
sigScreenPlay              ='SPLY'
sigSmoothie                ='Smoo'
sigStudio1                 ='ST/1'
```

```
sigStudio32              ='ST32'
sigStudio8               ='ST/8'
sigSwivel3D              ='SWVL'
sigVersaCad              ='VCAD'
```

Discussion
Constant names for creator codes are written as `sig` followed by the application name. Constant names for file types are written as `ft` followed by the `document` type.

Graphics Transfer Modes

Determine how images will be transferred.

```
// Boolean modes
// src modes are used with bitmaps and text;
// pat modes are used with lines and shapes
srcCopy                 =0
srcOr                   =1
srcXor                  =2
srcBic                  =3
notSrcCopy              =4
notSrcOr                =5
notSrcXor               =6
notSrcBic               =7
patCopy                 =8
patOr                   =9
patXor                  =10
patBic                  =11
notPatCopy              =12
notPatOr                =13
notPatXor               =14
notPatBic               =15
// Text dimming
grayishTextOr           =49
// Highlighting
hilite                  =50
hilitetransfermode      =50
// Arithmetic modes
blend                   =32
addPin                  =33
addOver                 =34
subPin                  =35
addMax                  =37
adMax                   =37
subOver                 =38
adMin                   =39
ditherCopy              =64
// Transparent mode
transparent             =36
```

`srcCopy`
: If the source is black, apply the foreground color to the destination; if the source is white, apply the background color; otherwise apply weighted portions of the foreground and background colors.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickdrawTypes.h`.

`srcOr`
: If the source is black, apply the foreground color to the destination; if the source is white, do nothing; otherwise apply weighted portions of the foreground color.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickdrawTypes.h`.

`srcXor`
: If the source is black, invert the destination (this operation is undefined for a colored destination). Otherwise, do nothing.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickdrawTypes.h`.

`srcBic`
: If the source is black, apply the background color to the destination. If the source is white, do nothing. Otherwise, apply weighted portions of the background color.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickdrawTypes.h`.

`notSrcCopy`
: If the source is white, apply the foreground color to the destination; if the source is black, apply the background color; otherwise apply weighted portions of the foreground and background colors.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickdrawTypes.h`.

`notSrcOr`
: If the source is white, apply the foreground color to the destination; if the source is black, do nothing; otherwise apply weighted portions of the foreground color.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickdrawTypes.h`.

`notSrcXor`
: If the source is white, invert the destination (this operation is undefined for a colored destination pixel). Otherwise, do nothing.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickdrawTypes.h`.

`notSrcBic`
: If the source is white, apply the background color to the destination. If the source is black, do nothing. Otherwise, apply weighted portions of the background color.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickdrawTypes.h`.

`patCopy`
: If the source is black, apply the foreground color to the destination; if the source is white, apply the background color; otherwise apply weighted portions of the foreground and background colors.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickdrawTypes.h`.

`patOr`
: If the source is black, apply the foreground color to the destination; if the source is white, do nothing; otherwise apply weighted portions of the foreground color.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickdrawTypes.h`.

`patXor`
: If the source is black, invert the destination (this operation is undefined for a colored destination). Otherwise, do nothing.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickdrawTypes.h`.

`patBic`
: If the source is black, apply the background color to the destination. If the source is white, do nothing. Otherwise, apply weighted portions of the background color.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickdrawTypes.h`.

`notPatCopy`
: If the source is white, apply the foreground color to the destination; if the source is black, apply the background color; otherwise apply weighted portions of the foreground and background colors.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickdrawTypes.h`.

`notPatOr`
: If the source is white, apply the foreground color to the destination; if the source is black, do nothing; otherwise apply weighted portions of the foreground color.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickdrawTypes.h`.

`notPatXor`
: If the source is white, invert the destination (this operation is undefined for a colored destination pixel). Otherwise, do nothing.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickdrawTypes.h`.

`grayishTextOr`
: Dim the destination. If in color, replace it with a blend of the foreground and background; if black-and-white, replace it with dithered black and white. This mode is used primarily for text.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickdrawTypes.h`.

`hilite`
: Replace the background color with the highlight color.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickdrawTypes.h`.

`hilitetransfermode`
: Replace the background color with the highlight color.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickdrawTypes.h`.

`blend`
: Replace the destination with a blend of the source and destination colors. If the destination is a bitmap, this is the same as srcCopy.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickdrawTypes.h`.

`addPin`
: Replace the destination with the sum of the source and destination, up to a maximum value. If the destination is a bitmap, this is the same as srcBic.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickdrawTypes.h`.

`addOver`
: Replace the destination with the sum of the source and destination, but if the resulting red, green, or blue value exceeds 65536, then subtract 65536 from it. If the destination is a bitmap, this is the same as srcXor.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickdrawTypes.h`.

`subPin`
: Replace the destination with the difference between the source and destination, but not less than a minimum value. If the destination is a bitmap, this is the same as srcOr.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickdrawTypes.h`.

`addMax`
: Compare the source and destination, and replace the destination with the greater value of each of the red, green, and blue components. If the destination is a bitmap, this is the same as srcBic.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickdrawTypes.h`.

`adMax`
: Compare the source and destination, and replace the destination with the greater value of each of the red, green, and blue components. If the destination is a bitmap, this is the same as srcBic.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickdrawTypes.h`.

`subOver`
: Replace the destination with the difference between the source and destination, but if the resulting red, green, or blue value is negative, then add 65536 to it. If the destination is a bitmap, this is the same as srcXor.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickdrawTypes.h`.

`adMin`
: Compare the source and destination, and replace the destination with the lesser value of each of the red, green, and blue components. If the destination is a bitmap, this is the same as srcOr.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickdrawTypes.h`.

`ditherCopy`
Replace the destination with a dither mix of the source and destination.

Available in Mac OS X v10.0 and later.

Declared in `QuickdrawTypes.h`.

`transparent`
Replace the destination with the source if the source is not equal to the background.

Available in Mac OS X v10.0 and later.

Declared in `QuickdrawTypes.h`.

See Also

For more information about graphics transfer modes, see *Inside Macintosh: Imaging With QuickDraw*.

Localization Codes

Identify languages, scripts, numbering systems, calendar systems, and geographical regions.

```
// Language codes:
langAfrikaans           =141   // smRoman script
langBreton              =142   // smRoman or smRoman/Celtic script
langAlbanian            =36    // smRoman script
langAmharic             =85    // smEthiopic script
langArabic              =12    // smArabic script
langArmenian            =51    // smArmenian script
langAssamese            =68    // smBengali script
langAymara              =134   // smRoman script
langAzerbaijanAr        =50    // Azerbaijani in smArabic script
langAzerbaijani         =49    // Azerbaijani in smCyrillic script
langBasque              =129   // smRoman script
langBelorussian         =46    // Synonym for langByelorussian
langUzbek               =47    // smCyrillic script
langBengali             =67    // smBengali script
langBulgarian           =44    // smCyrillic script
langBurmese             =77    // smBurmese script
langByelorussian        =46    // smCyrillic script
langCatalan             =130   // smRoman script
langChewa               =92    // synonym for langNyanja
langCroatian            =18    // modified smRoman/Croatian script
langCzech               =38    // smCentralEuroRoman script
langDanish              =7     // smRoman script
langDutch               =4     // smRoman script
langDzongkha            =137   // (Bhutan) smTibetan script
langEnglish             =0     // smRoman script
langEsperanto           =94    // smRoman script
langEstonian            =27    // smCentralEuroRoman script
langFaroese             =30    // smRoman/Icelandic script
langFarsi               =31    // modified smArabic/Farsi script
langFinnish             =13    // smRoman script
langFlemish             =34    // smRoman script
langFrench              =1     // smRoman script
langGalician            =140   // smRoman script
langGeorgian            =52    // smGeorgian script
langGerman              =2     // smRoman script
langGreek               =14    // Greek script using smRoman script
langGreekPoly           =148   // smGreek script
langGreenlandic         =149   // smRoman script
langGuarani             =133   // smRoman script
langGujarati            =69    // smGujarati script
langHebrew              =10    // smHebrew script
langHindi               =21    // smDevanagari script
langHungarian           =26    // smCentralEuroRoman script
langIcelandic           =15    // modified smRoman/Icelandic script
langIndonesian          =81    // smRoman script
langInuktitut           =143   // Inuit using smEthiopic script
langIrishGaelic         =35    // smRoman or smRoman/Celtic script
langIrishGaelicScript   =146   // smRoman/Gaelic script
langItalian             =3     // smRoman script
langJapanese            =11    // smJapanese script
langJavaneseRom         =138   // Javanese in smRoman script
langKannada             =73    // smKannada script
langKashmiri            =61    // smArabic script
langKazakh              =48    // smCyrillic script
langKhmer               =78    // smKhmer script
langKinyarwanda         =90    // smRoman script
langKirghiz             =54    // smCyrillic script
```

```
langKorean              =23    // smKorean script
langKurdish             =60    // smArabic script
langLao                 =79    // smLao script
langLatin               =131   // smRoman script
langLatvian             =28    // smCentralEuroRoman script
langLithuanian          =24    // smCentralEuroRoman script
langMacedonian          =43    // smCyrillic script
langMalagasy            =93    // smRoman script
langMalayalam           =72    // smMalayalam script
langMalayArabic         =84    // Malay in smArabic script
langMalayRoman          =83    // Malay in smRoman script
langMaltese             =16    // smRoman script
langManxGaelic          =145   // smRoman or smRoman/Celtic script
langMarathi             =66    // smDevanagari script
langMoldavian           =53    // smCyrillic script
langMongolian           =57    // Mongolian in smMongolian script
langMongolianCyr        =58    // Mongolian in smCyrillic script
langNepali              =64    // smDevanagari script
langNorwegian           =9     // smRoman script
langNyanja              =92    // smRoman script
langOriya               =71    // smOriya script
langOromo               =87    // smEthiopic script
langPashto              =59    // smArabic script
langPersian             =31    // Synonym for langFarsi
langPolish              =25    // smCentralEuroRoman script
langPortuguese          =8     // smRoman script
langPunjabi             =70    // smGurmukhi script
langQuechua             =132   // smRoman script
langRomanian            =37    // modified smRoman/Romanian script
langRuanda              =90    // synonym for langKinyarwanda
langRundi               =91    // smRoman script
langRussian             =32    // smCyrillic script
langSami                =29    // language of the Sami in Scandanavia
langSanskrit            =65    // smDevanagari script
langScottishGaelic      =144   // smRoman or smRoman/Celtic script
langSerbian             =42    // smCyrillic script
langSimpChinese         =33    // Mandarin in smSimpChinese script
langSindhi              =62    // smArabic script
langSinhalese           =76    // smSinhalese script
langSlovak              =39    // smCentralEuroRoman script
langSlovenian           =40    // modified smRoman/Croatian script
langSomali              =88    // smRoman script
langSpanish             =6     // smRoman script
langSundaneseRom        =139   // Sundanese in smRoman script
langSwahili             =89    // smRoman script
langSwedish             =5     // smRoman script
langTagalog             =82    // smRoman script
langTajiki              =55    // smCyrillic script
langTamil               =74    // smTamil script
langTatar               =135   // smCyrillic script
langTelugu              =75    // smTelugu script
langThai                =22    // smThai script
langTibetan             =63    // smTibetan script
langTigrinya            =86    // smEthiopic script
langTongan              =147   // smRoman script
langTradChinese         =19    // Mandarin in smTradChinese script
langTurkish             =17    // modified smRoman/Turkish script
langTurkmen             =56    // smCyrillic script
```

```
langUighur               =136   // smArabic script
langUkrainian            =45    // modified smCyrillic/Ukrainian script
langUrdu                 =20    // smArabic script
langVietnamese           =80    // smVietnamese script
langWelsh                =128   // modified smRoman/Celtic script
langYiddish              =41    // smHebrew script
langUnspecified          =32767
// Script codes
smArabic                 =4
smArmenian               =24
smBengali                =13
smBurmese                =19
smCentralEuroRoman       =29
smCyrillic               =7
smDevanagari             =9
smEthiopic               =28
smExtArabic              =31    // extended Arabic
smGeez                   =28    // Synonym for smEthiopic
smGeorgian               =23
smGreek                  =6
smGujarati               =11
smGurmukhi               =10
smHebrew                 =5
smJapanese               =1
smKannada                =16    // Kannada/Kanarese
smKhmer                  =20    // Khmer/Cambodian
smKorean                 =3
smLao                    =22
smMalayalam              =17
smMongolian              =27
smOriya                  =12
smRoman                  =0
smRSymbol                =8     // Right-left symbol
smSimpChinese            =25    // Simplified Chinese
smSinhalese              =18
smTamil                  =14
smTelugu                 =15
smThai                   =21
smTibetan                =26
smTradChinese            =2      // Traditional Chinese
smUnicodeScript          =0x7E   // Unicode
smUninterp               =32     // Uninterpreted symbols
smVietnamese             =30
// Calendar codes
calGregorian             =0
calArabicCivil           =1
calArabicLunar           =2
calJapanese              =3
calJewish                =4
calCoptic                =5
calPersian               =6
 // Integer format codes
intWestern               =0
intArabic                =1
intRoman                 =2
intJapanese              =3
intEuropean              =4
// Region codes
```

```
verAfrikaans             =102
verArabic                =16
verArmenian              =84
verAustralia             =15
verAustria               =92
verBengali               =60
verBhutan                =83
verBrazil                =71
verBreton                =77
verBritain               =2
verBulgaria              =72
verByeloRussian          =61
verCatalonia             =73
verChina                 =52
verCroatia               =68
verCyprus                =23
verCzech                 =56
verDenmark               =9
verEngCanada             =82
verEsperanto             =103
verEstonia               =44
verFarEastGeneric        =58
verFaroeIsl              =47
verFinland               =17
verFlemish               =6
verFrance                =1
verFrBelgium             =98
verFrCanada              =11
verFrenchUniversal       =91
verFrSwiss               =18
verGeorgian              =85
verGermany               =3
verGreece                =20
verGreecePoly            =40
verGreenland             =107
verGrSwiss               =19
verGujarati              =94
verHungary               =43
verIceland               =21
verIndiaHindi            =33
verIndiaUrdu             =96
verInternational         =37
verIran                  =48
verIreland               =50
verIrishGaelicScript     =81
verIsrael                =13
verItalianSwiss          =36
verItaly                 =4
verJapan                 =14
verKorea                 =51
verLatvia                =45
verLithuania             =41
verMacedonian            =67
verMagyar                =59
verMalta                 =22
verManxGaelic            =76
verMarathi               =104
verMultilingual          =74
```

`verNepal`	=106
`verNetherlands`	=5
`verNorway`	=12
`verNunavut`	=78
`verNynorsk`	=101
`verPakistanUrdu`	=34
`verPoland`	=42
`verPortugal`	=10
`verPunjabi`	=95
`verRomania`	=39
`verRussia`	=49
`verSami`	=46
`verScottishGaelic`	=75
`verScriptGeneric`	=55
`verSerbian`	=65
`verSingapore`	=100
`verSlovak`	=57
`verSlovenian`	=66
`verSpain`	=8
`verSpLatinAmerica`	=86
`verSweden`	=7
`verTaiwan`	=53
`verThailand`	=54
`verTibetan`	=105
`verTonga`	=88
`verTurkey`	=24
`verTurkishModified`	=35
`verUkraine`	=62
`verUS`	=0
`verUzbek`	=99
`verVietnam`	=97
`verWelsh`	=79

`langIrishGaelic`

Irish Gaelic for Ireland (without dot above).

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`verIreland`

Irish Gaelic for Ireland (without dot above).

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`langIrishGaelicScript`

Irish Gaelic for Ireland (using dot above).

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`verIrishGaelicScript`

Irish Gaelic for Ireland (using dot above).

Available in Mac OS X v10.0 and later.

Declared in `Script.h`.

`langSimpChinese`
Chinese using simplified characters.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smSimpChinese`
Chinese using simplified characters.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verChina`
Chinese using simplified characters.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`langTradChinese`
Chinese using traditional characters.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smTradChinese`
Chinese using traditional characters.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verTaiwan`
Chinese using traditional characters.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smCentralEuroRoman`
Script for Czech, Slovak, Polish, Hungarian, and the Baltic languages.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`smRSymbol`
Right-left symbol for bidirectional scripts (such as Arabic and Hebrew).
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verFarEastGeneric`
Generic Far East system (no language or script).
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verGreece`
Monotonic modern Greek.
Available in Mac OS X v10.0 and later.
Declared in `Script.h`.

`verGreecePoly`
: Polytonic ancient Greek.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verInternational`
: English for international use.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verMultilingual`
: No language or script.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verScriptGeneric`
: Generic script system (no language or script).

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verSpain`
: Spanish for Spain.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

`verSpLatinAmerica`
: Spanish for Latin America.

  Available in Mac OS X v10.0 and later.

  Declared in `Script.h`.

See Also

For more information about localization codes, see *Inside Macintosh: Text*. For general information about localization, see *Guide to Macintosh Software Localization* (Addison-Wesley 1992, ISBN 0-201-60856-1).

Other References

Component Creation Reference for QuickTime

Framework:	Frameworks/QuickTime.framework
Declared in	QuickTimeComponents.h

Overview

APIs are provided to help developer create new components that import and export data to and from QuickTime movies, including managing movie previews.

Functions by Task

Compressing Image Sequences

`SCCompressSequenceBegin` (page 517)
Initiates a sequence-compression operation.

`SCCompressSequenceEnd` (page 518)
Ends a sequence-compression operation.

`SCCompressSequenceFrame` (page 518)
Continues a sequence-compression operation.

Compressing Still Images

`SCCompressImage` (page 514)
Compresses an image that is stored in a PixMap structure.

`SCCompressPicture` (page 515)
Compresses a Picture structure that is stored by a handle.

`SCCompressPictureFile` (page 516)
Compresses a Picture structure that is stored in a file.

Configuring Movie Data Export Components

`MovieExportDoUserDialog` (page 468)
Requests that a component display its user dialog box.

`MovieExportSetProgressProc` (page 476)
Assigns a movie progress function.

Configuring Movie Data Import Components

`MovieImportDoUserDialog` (page 484)
Requests that a component display its user dialog box.

`MovieImportSetAuxiliaryData` (page 495)
Provides additional data to a component.

`MovieImportSetChunkSize` (page 496)
The amount of data a component works with at a time.

`MovieImportSetDimensions` (page 497)
Specifies a new track's spatial dimensions.

`MovieImportSetDuration` (page 498)
Controls the duration of the data that a component pastes into the target movie.

`MovieImportSetFromScrap` (page 499)
Indicates that the source data resides on the scrap.

`MovieImportSetMediaFile` (page 501)
Specifies a media file that is to receive the imported movie data.

`MovieImportSetProgressProc` (page 503)
Assigns a movie progress function.

`MovieImportSetSampleDescription` (page 504)
Provides a SampleDescription structure to a movie data import component.

`MovieImportSetSampleDuration` (page 505)
Sets the sample duration for new samples to be created with a component.

Creating a Compression Graphics World

`SCNewGWorld` (page 528)
Creates a graphics world based on the current compression settings.

Creating Previews

`PreviewMakePreview` (page 511)
Creates previews by allocating a handle to data that is to be added to a file.

`PreviewMakePreviewReference` (page 512)
Returns the type and identification number of a resource within a file to be used as the preview for a file.

Displaying Previews

`PreviewShowData` (page 513)
Displays a preview if it does not handle events.

Displaying the Standard Image-Compression Dialog Box

`SCRequestImageSettings` (page 530)
Displays the standard image dialog box to the user and shows default settings you have established.

`SCRequestSequenceSettings` (page 531)
Displays the standard sequence dialog box to the user and shows default settings you have established.

Exporting Movie Data

`MovieExportAddDataSource` (page 466)
Defines a data source for use with an export operation performed by MovieExportFromProceduresToDataRef.

`MovieExportDisposeGetDataAndPropertiesProcs` (page 467)
Disposes of the memory associated with the procedures returned by MovieExportNewGetDataAndPropertiesProcs.

`MovieExportFromProceduresToDataRef` (page 469)
Exports data provided by MovieExportAddDataSource to a specified location.

`MovieExportGetAuxiliaryData` (page 470)
Retrieves additional data from a component.

`MovieExportGetSettingsAsAtomContainer` (page 471)
Retrieves the current settings from the movie export component.

`MovieExportNewGetDataAndPropertiesProcs` (page 473)
Returns MovieExportGetPropertyProc and MovieExportGetDataProc callbacks that can be passed to MovieExportAddDataSource to create a new data source.

`MovieExportSetGetMoviePropertyProc` (page 475)
Specifies the procedure that the export component should call to retrieve movie level properties during MovieExportFromProceduresToDataRef.

`MovieExportSetSampleDescription` (page 476)
Requests the format of the exported data.

`MovieExportSetSettingsFromAtomContainer` (page 477)
Sets the movie export component's current configuration from passed settings data.

`MovieExportToDataRef` (page 479)
Allows an application to request that data be exported to a data reference instead of to a file.

`MovieExportToFile` (page 480)
Exports data to a file, using a movie data export component.

`MovieExportToHandle` (page 481)
Exports data from a movie, using a movie data export component.

`MovieExportValidate` (page 482)
Determines whether a movie export component can export all the data for a specified movie or track.

Exporting Text

`TextExportGetDisplayData` (page 546)
Retrieves text display information for the current sample in the specified text export component.

`TextExportGetSettings` (page 546)
    Retrieves the value of the text export option for the specified text export component.

`TextExportGetTimeFraction` (page 547)
    Retrieves the time scale the specified text export component uses to calculate time stamps.

`TextExportSetSettings` (page 548)
    Sets the value of the text export option for the specified text export component.

`TextExportSetTimeFraction` (page 548)
    Sets the time scale the specified text export component uses to calculate time stamps.

Getting Default Settings for an Image or a Sequence

`SCDefaultPictFileSettings` (page 522)
    Derives default compression settings for a Picture structure that is stored in a file.

`SCDefaultPictHandleSettings` (page 522)
    Derives default compression settings for a Picture structure that is stored by a handle.

`SCDefaultPixMapSettings` (page 523)
    Derives default compression settings for an image that is stored in a pixel map.

Handling Preview Events

`PreviewEvent` (page 511)
    May be called as appropriate if a preview component handles events.

Importing MIDI Files

`MIDIImportGetSettings` (page 464)
    Obtains settings that control the importation of MIDI files.

`MIDIImportSetSettings` (page 465)
    Define settings that control the importation of MIDI files.

Importing Movie Data

`MovieImportFile` (page 486)
    Imports data from a file, using a movie data import component.

`MovieImportGetAuxiliaryDataType` (page 488)
    Returns the type of the auxiliary data that a component can accept.

`MovieImportGetDestinationMediaType` (page 489)
    Returns the current type of a movie importer's destination media.

`MovieImportGetFileType` (page 490)
    Allows your movie data import component to tell the Movie Toolbox the appropriate file type for the most-recently imported movie file.

`MovieImportGetMIMETypeList` (page 492)
    Returns a list of MIME types supported by the movie import component.

`MovieImportGetSettingsAsAtomContainer` (page 493)
Retrieves the current settings from the movie import component.

`MovieImportHandle` (page 493)
Imports data from a handle, using a movie data import component.

`MovieImportSetOffsetAndLimit` (page 502)
Specifies location and size of data that should be imported.

`MovieImportSetOffsetAndLimit64` (page 503)
Specifies location and size of data that should be imported from a file.

`MovieImportSetSettingsFromAtomContainer` (page 505)
Sets the movie import component's current configuration from the passed settings data.

`MovieImportValidate` (page 506)
Allows your movie data import component to validate the data to be passed to your component.

`MovieImportValidateDataRef` (page 507)
Validates the data file indicated by the data reference.

Managing the Time

`ClockRateChanged` (page 458)
In a clock component, is called whenever the callback's time base rate changes.

`ClockSetTimeBase` (page 459)
In a clock component, is called when an application creates a time base that uses the clock component.

`ClockStartStopChanged` (page 459)
In a clock component, is called whenever the start or stop time of the callback's time base changes.

`ClockTimeChanged` (page 460)
In a clock component, is called whenever the callback's time base time value is set.

Movie Functions

`MovieImportSetNewMovieFlags` (page 501)
Implemented by a movie import component to determine the original flags for NewMovieFromDataRef.

Positioning Dialog Boxes and Rectangles

`SCGetBestDeviceRect` (page 524)
Determines the boundary rectangle that surrounds the display device that supports the largest color or grayscale palette.

`SCPositionDialog` (page 529)
Helps position a dialog box on the screen.

`SCPositionRect` (page 529)
Positions a rectangle on the screen.

Specifying a Test Image

`SCSetTestImagePictFile` (page 534)
Sets the dialog box's test image from a Picture structure that is stored in a picture file.

`SCSetTestImagePictHandle` (page 535)
Sets the dialog box's test image from a Picture structure that is stored in a handle.

`SCSetTestImagePixMap` (page 536)
Sets the dialog box's test image from a Picture structure that is stored in a PixMap structure.

Tween Component Requirements

`TweenerDoTween` (page 549)
Performs a tween operation.

`TweenerInitialize` (page 550)
Initializes your tween component for a single tween operation.

`TweenerReset` (page 551)
Cleans up when the tween operation is finished.

Using Callback Functions

`ClockCallMeWhen` (page 453)
In a clock component, schedules a callback event for invocation.

`ClockCancelCallBack` (page 454)
In a clock component, removes the specified callback event from the list of scheduled callback events for a time base.

`ClockDisposeCallBack` (page 455)
In a clock component, disposes of the memory associated with the specified callback event.

`ClockNewCallBack` (page 457)
In a clock component, allocates memory for a new callback event.

Working With Image or Sequence Settings

`SCGetInfo` (page 526)
Retrieves configuration information from the standard dialog component.

`SCSetInfo` (page 532)
Modifies the standard dialog component's configuration information.

Working With The Idle Manager

`MovieImportSetIdleManager` (page 499)
Lets a movie importer report its idling needs.

Working With the Timecode Media Handler

`TCFrameNumberToTimeCode` (page 538)
: Converts a frame number into its corresponding timecode time value.

`TCGetCurrentTimeCode` (page 538)
: Retrieves the timecode and source identification information for the current movie time.

`TCGetDisplayOptions` (page 539)
: Retrieves the text characteristics that apply to timecode information displayed in a movie.

`TCGetSourceRef` (page 540)
: Retrieves the source information from the timecode media sample reference.

`TCGetTimeCodeAtTime` (page 540)
: Returns a track's timecode information corresponding to a specific media time.

`TCGetTimeCodeFlags` (page 541)
: Retrieves the timecode control flags.

`TCSetDisplayOptions` (page 542)
: Sets the text characteristics that apply to timecode information displayed in a movie.

`TCSetSourceRef` (page 543)
: Changes the source information in the timecode media sample reference.

`TCSetTimeCodeFlags` (page 543)
: Changes the flag that affects how the toolbox handles timecode information.

`TCTimeCodeToFrameNumber` (page 544)
: Converts a timecode time value into its corresponding frame number.

`TCTimeCodeToString` (page 545)
: Converts a time value into a text string (HH:MM:SS:FF).

Supporting Functions

`ClockGetRate` (page 456)
: Fetches the rate of a specified clock.

`ClockGetRateChangeConstraints` (page 456)
: Obtains minimum and maximum delays that a clock could introduce during a rate change.

`ClockGetTime` (page 457)
: Obtains the current time according to a specified clock.

`DisposeMovieExportGetDataUPP` (page 461)
: Disposes of a MovieExportGetDataUPP pointer.

`DisposeMovieExportGetPropertyUPP` (page 461)
: Disposes of a MovieExportGetPropertyUPP pointer.

`DisposeMovieExportStageReachedCallbackUPP` (page 462)
: Disposes of a MovieExportStageReachedCallbackUPP pointer.

`DisposeSCModalFilterUPP` (page 462)
: Disposes of an SCModalFilterUPP pointer.

`DisposeSCModalHookUPP` (page 463)
: Disposes of an SCModalHookUPP pointer.

`GraphicsImageImportGetSequenceEnabled` (page 463)
    Undocumented

`GraphicsImageImportSetSequenceEnabled` (page 464)
    Undocumented

`MovieExportGetCreatorType` (page 470)
    Undocumented

`MovieExportGetFileNameExtension` (page 471)
    Undocumented

`MovieExportGetShortFileTypeString` (page 472)
    Undocumented

`MovieExportGetSourceMediaType` (page 473)
    Returns either the track type if a movie export component is track-specific or 0 if it is track-independent.

`MovieImportDataRef` (page 483)
    Undocumented

`MovieImportDoUserDialogDataRef` (page 485)
    Requests that a movie import component display its user dialog box.

`MovieImportEstimateCompletionTime` (page 486)
    Undocumented

`MovieImportGetDontBlock` (page 489)
    Undocumented

`MovieImportGetLoadState` (page 490)
    Undocumented

`MovieImportGetMaxLoadedTime` (page 491)
    Undocumented

`MovieImportGetSampleDescription` (page 492)
    Gets the current sample description for a movie import component.

`MovieImportIdle` (page 495)
    Undocumented

`MovieImportSetDontBlock` (page 497)
    Undocumented

`MovieImportSetMediaDataRef` (page 500)
    Specifies a storage location that is to receive imported movie data.

`NewMovieExportGetDataUPP` (page 508)
    Allocates a Universal Procedure Pointer for the MovieExportGetDataProc callback.

`NewMovieExportGetPropertyUPP` (page 509)
    Allocates a Universal Procedure Pointer for the MovieExportGetPropertyProc callback.

`NewMovieExportStageReachedCallbackUPP` (page 509)
    Allocates a new Universal Procedure Pointer for a MovieExportStageReachedCallbackProc callback.

`NewSCModalFilterUPP` (page 510)
    Allocates a Universal Procedure Pointer for the SCModalFilterProc callback.

`NewSCModalHookUPP` (page 510)
    Allocates a Universal Procedure Pointer for the SCModalHookProc callback.

`SCAsyncIdle` (page 514)
: Called occasionally while performing asynchronous compression with SCCompressSequenceFrameAsync.

`SCAudioInvokeLegacyCodecOptionsDialog` (page 514)
: Invokes the legacy code options dialog of an audio codec component.

`SCCompressSequenceFrameAsync` (page 520)
: An asynchronous variant of SCCompressSequenceFrame, with a completion callback.

`SCCopyCompressionSessionOptions` (page 521)
: Creates a compression session options object based upon the settings in the Standard Compression component.

`SCGetCompressFlags` (page 524)
: Gets compression flags for a standard image-compression dialog component.

`SCGetCompressionExtended` (page 525)
: Undocumented

`SCGetSettingsAsAtomContainer` (page 527)
: Places the current configuration from the standard image-compression component in a QT atom container.

`SCGetSettingsAsText` (page 527)
: Undocumented

`SCSetCompressFlags` (page 532)
: Sets compression flags for a standard image-compression dialog component.

`SCSetSettingsFromAtomContainer` (page 533)
: Sets the standard image-compression component's current configuration from data in a QT atom container.

Functions

ClockCallMeWhen

In a clock component, schedules a callback event for invocation.

```
ComponentResult ClockCallMeWhen (
   ComponentInstance aClock,
   QTCallBack cb,
   long param1,
   long param2,
   long param3
);
```

Parameters

aClock
: Specifies the clock for the operation. Applications obtain this identifier from `OpenComponent`.

cb
: Specifies the callback event for the operation. The Movie Toolbox obtains this value from your component's `ClockNewCallBack` (page 457) function.

param1
Contains data supplied to the Movie Toolbox in the param1 parameter to the `CallMeWhen` (page 179) function. Your component interprets this parameter based on the value of the `callBackType` parameter to the `ClockNewCallBack` (page 457) function. If `callBackType` is set to `callBackAtTime`, the param1 parameter contains flags (see below) indicating when to invoke your callback function for this callback event. If the `callBackType` parameter is set to `callBackAtRate`, param1 contains flags (see below) indicating when to invoke your callback function for this event.

param2
Contains data supplied to the Movie Toolbox in the param2 parameter to the `CallMeWhen` (page 179) function. Your component interprets this parameter based on the value of the `callBackType` parameter to the `ClockNewCallBack` (page 457) function. If `callBackType` is set to `callBackAtTime`, the param2 parameter contains the time value at which your callback function is to be invoked for this event. The param1 parameter contains flags affecting when the Movie Toolbox calls your function. If `callBackType` is set to `callBackAtRate`, the param2 parameter contains the rate value at which your callback function is to be invoked for this event.

param3
Contains data supplied to the Movie Toolbox in the param3 parameter to the `CallMeWhen` (page 179) function. If `cbType` is set to `callBackAtTime`, param3 contains the time scale in which to interpret the time value that is stored in param2.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
If your clock component successfully schedules the callback event, you should call the `AddCallBackToTimeBase` (page 178) function to add it to the list of callback events for the corresponding time base. If your component cannot schedule the callback event, it should return an appropriate error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

ClockCancelCallBack

In a clock component, removes the specified callback event from the list of scheduled callback events for a time base.

```
ComponentResult ClockCancelCallBack (
   ComponentInstance aClock,
   QTCallBack cb
);
```

Parameters
aClock
Specifies the clock for the operation. Your application obtains this identifier from the Component Manager's `OpenComponent` function.

cb

Specifies the callback event for the operation. The Movie Toolbox obtains this value from your component's `ClockNewCallBack` (page 457) function.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

If your clock component successfully cancels the callback event, you should call the `RemoveCallBackFromTimeBase` (page 284) function so that the Movie Toolbox can remove the callback event from its list of scheduled events.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

ClockDisposeCallBack

In a clock component, disposes of the memory associated with the specified callback event.

```
ComponentResult ClockDisposeCallBack (
   ComponentInstance aClock,
   QTCallBack cb
);
```

Parameters

aClock

Specifies the clock for the operation. Applications obtain this identifier from the Component Manager's `OpenComponent` function.

cb

Specifies the callback event for the operation. The Movie Toolbox obtains this value from your component's `ClockNewCallBack` (page 457) function.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

You should not call this function at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

ClockGetRate

Fetches the rate of a specified clock.

```
ComponentResult ClockGetRate (
   ComponentInstance aClock,
   Fixed *rate
);
```

Parameters

aClock

Specifies the clock for the operation. Applications obtain this identifier from the Component Manager's `OpenComponent` function.

rate

Pointer to memory where the clock rate is returned.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

ClockGetRateChangeConstraints

Obtains minimum and maximum delays that a clock could introduce during a rate change.

```
ComponentResult ClockGetRateChangeConstraints (
   ComponentInstance aClock,
   TimeRecord *minimumDelay,
   TimeRecord *maximumDelay
);
```

Parameters

aClock

Specifies the clock for the operation. Applications obtain this identifier from `OpenComponent`.

minimum

A pointer to a `TimeRecord` structure that the clock will update with the minimum delay introduced during a rate change. You can pass `NIL` if you do not want to receive this information.

maximum

A pointer to a `TimeRecord` structure that the clock will update with the maximum delay introduced during a rate change. You can pass `NIL` if you do not want to receive this information.

Return Value

See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error. Returns `badComponentSelector` if the component does not support the call.

Version Notes

Introduced in QuickTime 6.4.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`QuickTimeComponents.h`

ClockGetTime

Obtains the current time according to a specified clock.

```
ComponentResult ClockGetTime (
   ComponentInstance aClock,
   TimeRecord *out
);
```

Parameters

aClock

Specifies the clock for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`.

out

A pointer to a `TimeRecord` structure. The clock component updates this structure with the current time information. Specifically, the clock component sets the `value` field and the `scale` field in the time structure. Your clock component should always return values in its native time scale. This time scale does not change during the life of the component connection.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

ClockNewCallBack

In a clock component, allocates memory for a new callback event.

```
QTCallBack ClockNewCallBack (
   ComponentInstance aClock,
   TimeBase tb,
   short callBackType
);
```

Parameters

aClock

Specifies the clock for the operation. Applications obtain this identifier from the Component Manager's `OpenComponent` function.

tb

Specifies the callback event's time base. Typically, your component does not need to save this specification. You can use the Movie Toolbox's `GetCallBackTimeBase` (page 203) function to determine the callback event's time base when it is invoked. For more information about time bases, see *Inside Macintosh: QuickTime*.

callBackType

Contains a constant (see below) that specifies when the callback event is to be invoked. The value of this parameter governs how your component interprets the data supplied in the param1, param2, and param3 parameters to `ClockCallMeWhen` (page 453). See these constants:

`callBackAtTime`

`callBackAtRate`

`callBackAtTimeJump`

`callBackAtInterrupt`

Return Value

A pointer to a `CallBackRecord` structure. Your software can pass this structure to other functions, such as `ClockRateChanged` (page 458).

Discussion

Your component allocates the memory required to support the callback event. The memory must be in a locked block and must begin with a `QTCallBackHeader` structure initialized to 0. Your component can allocate an arbitrarily large piece of memory for the callback event.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

ClockRateChanged

In a clock component, is called whenever the callback's time base rate changes.

```
ComponentResult ClockRateChanged (
   ComponentInstance aClock,
   QTCallBack cb
);
```

Parameters

aClock

Specifies the clock for the operation. Applications obtain this identifier from the Component Manager's `OpenComponent` function.

cb

Specifies the callback for the operation. The Movie Toolbox obtains this value from your component's `ClockNewCallBack` (page 457) function.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion
The Movie Toolbox calls this function once for each qualified callback function associated with the time base. Note that the Movie Toolbox calls this function only for callback events that are currently scheduled.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

ClockSetTimeBase

In a clock component, is called when an application creates a time base that uses the clock component.

```
ComponentResult ClockSetTimeBase (
   ComponentInstance aClock,
   TimeBase tb
);
```

Parameters

`aClock`
: Specifies the clock for the operation. Applications obtain this identifier from the Component Manager's `OpenComponent` function.

`tb`
: Specifies the time base that is associated with the clock.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

ClockStartStopChanged

In a clock component, is called whenever the start or stop time of the callback's time base changes.

```
ComponentResult ClockStartStopChanged (
   ComponentInstance aClock,
   QTCallBack cb,
   Boolean startChanged,
   Boolean stopChanged
);
```

Parameters

aClock

Specifies the clock for the operation. Applications obtain this identifier from the Component Manager's `OpenComponent` function.

cb

Specifies the callback for the operation. The Movie Toolbox obtains this value from your component's `ClockNewCallBack` (page 457) function.

startChanged

Indicates that the start time of the time base associated with the clock component instance has changed.

stopChanged

Indicates that the stop time of the time base associated with the clock component instance has changed.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The Movie Toolbox calls this function once for each qualified callback function associated with the time base. Note that the Movie Toolbox calls this function only for callback events that are currently scheduled.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

ClockTimeChanged

In a clock component, is called whenever the callback's time base time value is set.

```
ComponentResult ClockTimeChanged (
   ComponentInstance aClock,
   QTCallBack cb
);
```

Parameters

aClock

Specifies the clock for the operation. Applications obtain this identifier from the Component Manager's `OpenComponent` function.

cb

Specifies the callback for the operation. The Movie Toolbox obtains this value from your component's `ClockNewCallBack` (page 457) function.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

DisposeMovieExportGetDataUPP

Disposes of a MovieExportGetDataUPP pointer.

```
void DisposeMovieExportGetDataUPP (
   MovieExportGetDataUPP userUPP
);
```

Parameters

`userUPP`
   A `MovieExportGetDataUPP` pointer. See `Universal Procedure Pointers`.

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CIVideoDemoGL

qtmoviefromprocs

qtmoviefromprocs.win

Declared In
`QuickTimeComponents.h`

DisposeMovieExportGetPropertyUPP

Disposes of a MovieExportGetPropertyUPP pointer.

```
void DisposeMovieExportGetPropertyUPP (
   MovieExportGetPropertyUPP userUPP
);
```

Parameters

`userUPP`
   A `MovieExportGetPropertyUPP` pointer. See `Universal Procedure Pointers`.

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CIVideoDemoGL

qtmoviefromprocs

qtmoviefromprocs.win

Declared In
`QuickTimeComponents.h`

DisposeMovieExportStageReachedCallbackUPP

Disposes of a MovieExportStageReachedCallbackUPP pointer.

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

Parameters

userUPP
    A `MovieExportStageReachedCallbackUPP` pointer.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`QuickTimeComponents.h`

DisposeSCModalFilterUPP

Disposes of an SCModalFilterUPP pointer.

```
void DisposeSCModalFilterUPP (
   SCModalFilterUPP userUPP
);
```

Parameters

userUPP
    An `SCModalFilterUPP` pointer. See `Universal Procedure Pointers`.

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qtcompress

qtcompress.win

Declared In
`QuickTimeComponents.h`

DisposeSCModalHookUPP

Disposes of an SCModalHookUPP pointer.

```
void DisposeSCModalHookUPP (
   SCModalHookUPP userUPP
);
```

Parameters

`userUPP`
An `SCModalHookUPP` pointer. See `Universal Procedure Pointers`.

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qtcompress

qtcompress.win

Declared In
`QuickTimeComponents.h`

GraphicsImageImportGetSequenceEnabled

Undocumented

```
ComponentResult GraphicsImageImportGetSequenceEnabled (
   GraphicImageMovieImportComponent ci,
   Boolean *enable
);
```

Parameters

`ci`
The component instance that identifies your connection to the movie importer component.

`enable`
A pointer to a Boolean that returns TRUE if enabled, FALSE otherwise.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

GraphicsImageImportSetSequenceEnabled

Undocumented

```
ComponentResult GraphicsImageImportSetSequenceEnabled (
   GraphicImageMovieImportComponent ci,
   Boolean enable
);
```

Parameters

ci

The component instance that identifies your connection to the movie importer component.

enable

Pass TRUE to enable, FALSE to disable.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Graphic Import-Export

ImproveYourImage

Declared In
`QuickTimeComponents.h`

MIDIImportGetSettings

Obtains settings that control the importation of MIDI files.

```
ComponentResult MIDIImportGetSettings (
   TextExportComponent ci,
   long *setting
);
```

Parameters

ci

A text export component instance used to import a MIDI file. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

setting

Flags (see below) that control the importation of MIDI files. The flags correspond to the checkboxes in the MIDI Import Options dialog box. See these constants:

`kMIDIImportSilenceBefore`

`kMIDIImportSilenceAfter`

`kMIDIImport20Playable`

`kMIDIImportWantLyrics`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

MIDIImportSetSettings

Define settings that control the importation of MIDI files.

```
ComponentResult MIDIImportSetSettings (
   TextExportComponent ci,
   long setting
);
```

Parameters

ci

A text export component instance used to import a MIDI file. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

setting

Flags (see below) that control the importation of MIDI files. The flags correspond to the checkboxes in the MIDI Import Options dialog box. See these constants:

`kMIDIImportSilenceBefore`

`kMIDIImportSilenceAfter`

`kMIDIImport20Playable`

`kMIDIImportWantLyrics`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

MovieExportAddDataSource

Defines a data source for use with an export operation performed by MovieExportFromProceduresToDataRef.

```
ComponentResult MovieExportAddDataSource (
   MovieExportComponent ci,
   OSType trackType,
   TimeScale scale,
   long *trackID,
   MovieExportGetPropertyUPP getPropertyProc,
   MovieExportGetDataUPP getDataProc,
   void *refCon
);
```

Parameters

ci
: A movie export component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

trackType
: The type of media provided by this data source. This normally corresponds to a QuickTime media type such as `VideoMediaType` or `SoundMediaType`.

scale
: The time scale for time values passed to `getDataProc` parameter. If the source data is being taken from a QuickTime track, this value is typically the media's time scale.

trackID
: An identifier for the data source. This identifier is returned from the call.

getPropertyProc
: A `MovieExportGetPropertyProc` callback that provides information about processing source samples.

getDataProc
: A `MovieExportGetDataProc` callback the export component uses to request sample data.

refCon
: Passed to the procedures specified in the `getPropertyProc` and `getDataProc` parameters. Use this parameter to point to a data structure containing any information your callbacks need.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Before starting an export operation, all the data sources must be defined by calling this function once for each data source.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CIVideoDemoGL
ElectricImageComponent
ElectricImageComponent.win
qtmoviefromprocs
qtmoviefromprocs.win

Declared In
`QuickTimeComponents.h`

MovieExportDisposeGetDataAndPropertiesProcs

Disposes of the memory associated with the procedures returned by MovieExportNewGetDataAndPropertiesProcs.

```
ComponentResult MovieExportDisposeGetDataAndPropertiesProcs (
   MovieExportComponent ci,
   MovieExportGetPropertyUPP getPropertyProc,
   MovieExportGetDataUPP getDataProc,
   void *refCon
);
```

Parameters

ci
A movie export component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

getPropertyProc
A `MovieExportGetPropertyProc` callback that provides information about processing source samples.

getDataProc
A `MovieExportGetDataProc` callback that the export component uses to request sample data.

refCon
Passed to the procedures specified in the `getPropertyProc` and `getDataProc` parameters. Use this parameter to point to a data structure containing any information your callbacks need.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CIVideoDemoGL

ElectricImageComponent

ElectricImageComponent.win

Declared In

`QuickTimeComponents.h`

MovieExportDoUserDialog

Requests that a component display its user dialog box.

```
ComponentResult MovieExportDoUserDialog (
   MovieExportComponent ci,
   Movie theMovie,
   Track onlyThisTrack,
   TimeValue startTime,
   TimeValue duration,
   Boolean *canceled
);
```

Parameters

ci

A movie export component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

theMovie

The movie containing the data to be exported.

onlyThisTrack

Specifies that the export component should only attempt to export the data from a single track. If this parameter is set to `NIL`, the exporter should attempt to export the entire movie, or all of the tracks in the movie that it can export. For example, an audio export component might export multiple audio tracks, mixing them if necessary. If this parameter is not `NIL`, the exporter should attempt to export only the specified track.

startTime

The movie time at which to begin the export operation. If you pass 0, the operation should start at the beginning of the movie or track.

duration

The duration, in movie timescale units, of the segment to be exported. To export the entire movie, or an entire track, pass in the value returned by `GetMovieDuration` (page 211) or `GetTrackDuration` (page 1607), minus the value passed in `startTime`, as described above.

canceled

A pointer to a Boolean value. Your component should set this value to TRUE if the user cancels the dialog box, otherwise FALSE. If the user cancels the dialog box, your component should revert to its settings as they were before executing this function.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

ImportExportMovie

qtdataexchange.win

qthintmovies.win

qtmoviefromprocs.win

ThreadsExportMovie

Declared In

`QuickTimeComponents.h`

MovieExportFromProceduresToDataRef

Exports data provided by MovieExportAddDataSource to a specified location.

```
ComponentResult MovieExportFromProceduresToDataRef (
   MovieExportComponent ci,
   Handle dataRef,
   OSType dataRefType
);
```

Parameters

ci

A movie export component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

dataRef

The data reference for the export operation.

dataRefType

The type identifier for the data reference specified by `dataRef`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function exports data provided by `MovieExportAddDataSource` (page 466) to a location specified by `dataRef` and `dataRefType`. Typically `dataRef` contains a Macintosh file alias and `dataRefType` is set to `rAliasType`.

Special Considerations

Movie data export components that support export operations from procedures must set the `canMovieExportFromProcedures` flag in their component flags.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CIVideoDemoGL

ElectricImageComponent

ElectricImageComponent.win

qtmoviefromprocs

qtmoviefromprocs.win

Declared In
`QuickTimeComponents.h`

MovieExportGetAuxiliaryData

Retrieves additional data from a component.

```
ComponentResult MovieExportGetAuxiliaryData (
   MovieExportComponent ci,
   Handle dataH,
   OSType *handleType
);
```

Parameters

ci
A movie export component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

dataH
A handle that is to be filled with the additional data. Your component should resize this handle as appropriate. Your component is not responsible for disposing of this handle.

handleType
A pointer to the type of data you place in the handle specified by the `data` parameter.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Your component should expect the application to call this function after the export process ends.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

MovieExportGetCreatorType

Undocumented

```
ComponentResult MovieExportGetCreatorType (
   MovieExportComponent ci,
   OSType *creator
);
```

Parameters

ci
A movie export component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

creator

Undocumented

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

MovieExportGetFileNameExtension

Undocumented

```
ComponentResult MovieExportGetFileNameExtension (
   MovieExportComponent ci,
   OSType *extension
);
```

Parameters

ci

A movie export component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

extension

Undocumented

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

MovieExportGetSettingsAsAtomContainer

Retrieves the current settings from the movie export component.

```
ComponentResult MovieExportGetSettingsAsAtomContainer (
   MovieExportComponent ci,
   QTAtomContainer *settings
);
```

Parameters

ci

A movie export component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

settings

The address where the newly-created atom container should be stored by the call. The caller is responsible for disposing of the returned QT atom container.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Applications can call this function to obtain a correctly formatted atom container to use with `MovieExportSetSettingsFromAtomContainer` (page 477). This might be done after a call to `MovieExportDoUserDialog` (page 468), for example, to apply the user-obtained settings to a series of exports.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

BackgroundExporter

qtdataexchange

qtdataexchange.win

qtmoviefromprocs.win

ThreadsExportMovie

Declared In

`QuickTimeComponents.h`

MovieExportGetShortFileTypeString

Undocumented

```
ComponentResult MovieExportGetShortFileTypeString (
   MovieExportComponent ci,
   Str255 typeString
);
```

Parameters

ci

A movie export component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

typeString

Undocumented

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

MovieExportGetSourceMediaType

Returns either the track type if a movie export component is track-specific or 0 if it is track-independent.

```
ComponentResult MovieExportGetSourceMediaType (
   MovieExportComponent ci,
   OSType *mediaType
);
```

Parameters

ci
: A movie export component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

mediaType
: The track type if the component is track-specific or 0 if it is track-independent.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This routine returns the same values that were previously stored in the `componentManufacturer` field of the `ComponentDescription` structure. This frees up the field to be used for the manufacturer.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

MovieExportNewGetDataAndPropertiesProcs

Returns MovieExportGetPropertyProc and MovieExportGetDataProc callbacks that can be passed to MovieExportAddDataSource to create a new data source.

```
ComponentResult MovieExportNewGetDataAndPropertiesProcs (
   MovieExportComponent ci,
   OSType trackType,
   TimeScale *scale,
   Movie theMovie,
   Track theTrack,
   TimeValue startTime,
   TimeValue duration,
   MovieExportGetPropertyUPP *getPropertyProc,
   MovieExportGetDataUPP *getDataProc,
   void **refCon
);
```

Parameters

ci

A movie export component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

trackType

The format of the data to be generated by the returned `MovieExportGetDataProc`.

scale

The time scale returned from this function; this should be passed on to `MovieExportAddDataSource` (page 466) with the procedures.

theMovie

The movie for this operation, supplied by the Movie Toolbox. Your component may use this identifier to obtain sample data from the movie or to obtain information about the movie.

theTrack

The track for this operation. This track identifier is supplied by the Movie Toolbox.

startTime

The starting point of the track or movie segment to be converted. This time value is expressed in the movie's time coordinate system.

duration

The duration of the track or movie segment to be converted. This duration value is expressed in the movie's time coordinate system.

getPropertyProc

A `MovieExportGetPropertyProc` callback that provides information about processing source samples.

getDataProc

A `MovieExportGetDataProc` callback that the export component uses to request sample data.

refCon

Passed to the procedures specified in the `getPropertyProc` and `getDataProc` parameters. Use this parameter to point to a data structure containing any information your callbacks need.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function exists in order to provide a standard way of getting data using this protocol out of a movie or track. The returned procedures must be disposed by calling `MovieExportDisposeGetDataAndPropertiesProcs` (page 467).

Special Considerations

This function is only implemented by movie data export components.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CIVideoDemoGL

ElectricImageComponent

ElectricImageComponent.win

Declared In
`QuickTimeComponents.h`

MovieExportSetGetMoviePropertyProc

Specifies the procedure that the export component should call to retrieve movie level properties during MovieExportFromProceduresToDataRef.

```
ComponentResult MovieExportSetGetMoviePropertyProc (
   MovieExportComponent ci,
   MovieExportGetPropertyUPP getPropertyProc,
   void *refCon
);
```

Parameters

ci

A movie export component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

getPropertyProc

The `MovieExportGetPropertyProc` callback that the export component will call to retrieve movie-level properties.

refCon

The reference value that will be passed to the callback specified by `getPropertyProc`. Use this parameter to point to a data structure containing any information your callback needs.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4. With QuickTime 4, applications can specify a `MovieExportGetPropertyProc` that will be called to retrieve movie level properties during the exporter's `MovieExportFromProceduresToDataRef` (page 469) execution. This procedure is identical to a data source property procedure except that it is called for movie properties. For example, with QuickTime 4, the QuickTime movie export component calls the procedure to retrieve the time scale for the exported movie. If the property procedure is not specified or doesn't support this property, than the default movie time scale (600) is used.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

MovieExportSetProgressProc

Assigns a movie progress function.

```
ComponentResult MovieExportSetProgressProc (
   MovieExportComponent ci,
   MovieProgressUPP proc,
   long refcon
);
```

Parameters

ci
: A movie export component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

proc
: A pointer to the application's `MovieProgressProc` callback. If this parameter is set to `NIL`, the application is removing its progress function. In this case, your component should stop calling the progress function.

refcon
: A reference constant. Your component should pass this constant back to the application's progress function whenever you call that function. Use this parameter to point to a data structure containing any information the callback needs.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
These progress functions must support the same interface as Movie Toolbox progress functions. Note that this interface not only allows you to report progress to the application, but also allows the application to cancel the request.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CIVideoDemoGL

ThreadsExportMovie

Declared In
`QuickTimeComponents.h`

MovieExportSetSampleDescription

Requests the format of the exported data.

```
ComponentResult MovieExportSetSampleDescription (
   MovieExportComponent ci,
   SampleDescriptionHandle desc,
   OSType mediaType
);
```

Parameters

ci

A movie export component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

desc

A handle to a valid `SampleDescription` structure.

mediaType

The type of media the `SampleDescription` structure is for. For example, if the sample description was a sound description, this parameter would be set to `SoundMediaType`.

Return Value

See `Error Codes`. Returns `badComponentSelector` if you should be passing a QT atom container (see discussion, below). Returns `noErr` if there is no error.

Discussion

A movie export component may use all, some, or none of the settings from the `SampleDescription` structure.

If your application attempts to set the sample description using this function, and receives the `badComponentSelector` error, you may need to pass in the sample description using `MovieExportSetSettingsFromAtomContainer` (page 477). You can use `MovieExportGetSettingsAsAtomContainer` (page 471) to obtain a correctly formatted atom container to modify.

Special Considerations

This function is not implemented by all movie export components, but is supported by the sound movie export component, for example.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

MovieToAIFF

soundsnippets

soundsnippets.win

Declared In

`QuickTimeComponents.h`

MovieExportSetSettingsFromAtomContainer

Sets the movie export component's current configuration from passed settings data.

```
ComponentResult MovieExportSetSettingsFromAtomContainer (
   MovieExportComponent ci,
   QTAtomContainer settings
);
```

Parameters

ci

A movie export component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

settings

A QT atom container that contains the settings.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The atom container may contain atoms other than those expected by the particular component type or may be missing certain atoms. This function uses only those settings it understands.

Here is sample code that overrides compression settings:

```
// MovieExportSetSettingsFromAtomContainer coding example
ComponentInstance sc;
QTAtomContainer compressorData;
SCSpatialSettings ss;
sc =OpenDefaultComponent(StandardCompressionType,
                              StandardCompressionSubType);
ss.codecType =kCinepakCodecType;
ss.codec =NIL;
ss.depth =0;
ss.spatialQuality =codecHighQuality
err =SCSetInfo(sc, scSpatialSettingsType, &ss);
err =SCGetSettingsAsAtomContainer(sc, &compressorData);
MovieExportSetSettingsFromAtomContainer (qtvrExport, compressorData);
```

Special Considerations

Some movie export components treat sample descriptions as part of their settings. If your application attempts to set the sample description using `MovieExportSetSampleDescription` (page 476), and receives the `badComponentSelector` error, you may need to pass in the `SampleDescription` structure using this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

BackgroundExporter

qtdataexchange

qtdataexchange.win

ThreadsExportMovie

vrmakepano

Declared In
`QuickTimeComponents.h`

MovieExportToDataRef

Allows an application to request that data be exported to a data reference instead of to a file.

```
ComponentResult MovieExportToDataRef (
   MovieExportComponent ci,
   Handle dataRef,
   OSType dataRefType,
   Movie theMovie,
   Track onlyThisTrack,
   TimeValue startTime,
   TimeValue duration
);
```

Parameters

ci
A movie export component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

dataRef
A handle to a data reference indicating where the data should be stored.

dataRefType
The type of the data reference. For exporting to a file, the `dataRef` is a Macintosh file alias and the `dataRefType` is `rAliasType`.

theMovie
The movie for this operation. This movie identifier is supplied by the Movie Toolbox. Your component may use this identifier to obtain sample data from the movie or to obtain information about the movie.

onlyThisTrack
Identifies a track that is to be converted. This track identifier is supplied by the Movie Toolbox. If this parameter contains a track identifier, your component must convert only the specified track.

startTime
The starting point of the track or movie segment to be converted. This time value is expressed in the movie's time coordinate system.

duration
The duration of the track or movie segment to be converted. This duration value is expressed in the movie's time coordinate system.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CIVideoDemoGL
ElectricImageComponent

ElectricImageComponent.win

ThreadsExportMovie

Declared In
`QuickTimeComponents.h`

MovieExportToFile

Exports data to a file, using a movie data export component.

```
ComponentResult MovieExportToFile (
   MovieExportComponent ci,
   const FSSpec *theFile,
   Movie theMovie,
   Track onlyThisTrack,
   TimeValue startTime,
   TimeValue duration
);
```

Parameters

ci

A movie export component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

theFile

A pointer to the file that is to receive the converted movie data. This file's type value corresponds to your component's subtype value.

theMovie

The movie for this operation. This movie identifier is supplied by the Movie Toolbox. Your component may use this identifier to obtain sample data from the movie or to obtain information about the movie.

onlyThisTrack

Identifies a track that is to be converted. This track identifier is supplied by the Movie Toolbox. If this parameter contains a track identifier, your component must convert only the specified track.

startTime

The starting point of the track or movie segment to be converted. This time value is expressed in the movie's time coordinate system.

duration

The duration of the track or movie segment to be converted. This duration value is expressed in the movie's time coordinate system.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
The requesting program or Movie Toolbox must create the destination file before calling this function. Furthermore, your component may not destroy any data in the destination file. If you cannot add data to the specified file, return an appropriate error. If your component can write data to a file, be sure to set the `canMovieExportFiles` flag in the `componentFlags` field of your component's `ComponentDescription` structure. Here is an example of using this function with a flattener component:

```
// MovieExportToFile coding example
ComponentDescription desc;
```

```
Component flattener;
ComponentInstance qtvrExport =NIL;
desc.componentType =MovieExportType;
desc.componentSubType =MovieFileType;
desc.componentManufacturer =QTVRFlattenerType;
flattener =FindNextComponent(NIL, &desc);
if (flattener) qtvrExport =OpenComponent (flattener);
if (qtvrExport)
    MovieExportToFile (qtvrExport, &myFileSpec, myQTVRMovie, NIL, 0, 0);
```

Special Considerations

Your component must be prepared to perform this function at any time. You should not expect that any of your component's configuration functions will be called first.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
vrmakepano

VRMakePano Library

Declared In
`QuickTimeComponents.h`

MovieExportToHandle

Exports data from a movie, using a movie data export component.

```
ComponentResult MovieExportToHandle (
   MovieExportComponent ci,
   Handle dataH,
   Movie theMovie,
   Track onlyThisTrack,
   TimeValue startTime,
   TimeValue duration
);
```

Parameters

ci

A movie export component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

dataH

A handle to be filled with the converted movie data. Your component must write data into this handle that corresponds to your component's subtype value. Your component should resize this handle as appropriate.

theMovie

The movie for this operation. This movie identifier is supplied by the Movie Toolbox. Your component may use this identifier to obtain sample data from the movie or to obtain information about the movie.

onlyThisTrack
Identifies a track that is to be converted. This track identifier is supplied by the Movie Toolbox. If this parameter contains a track identifier, your component must convert only the specified track.

startTime
The starting point of the track or movie segment to be converted. This time value is expressed in the movie's time coordinate system.

duration
The duration of the track or movie segment to be converted. This duration value is expressed in the movie's time coordinate system.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Your component must be prepared to perform this function at any time. You should not expect that any of your component's configuration functions will be called first. If your component can write data to a handle, be sure to set the `canMovieExportHandles` flag in in the `componentFlags` field of your component's `ComponentDescription` structure.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

MovieExportValidate

Determines whether a movie export component can export all the data for a specified movie or track.

```
ComponentResult MovieExportValidate (
   MovieExportComponent ci,
   Movie theMovie,
   Track onlyThisTrack,
   Boolean *valid
);
```

Parameters

ci
A movie export component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

theMovie
The movie to validate.

onlyThisTrack
A track within the movie to validate, or `NIL` if the entire movie is to be validated.

valid
A pointer to a Boolean value. If the data for the movie or track can be exported by the component, the value is TRUE.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function allows an application to determine if a particular movie or track could be exported by the specified movie data export component. The movie or track is passed in the `theMovie` and `onlyThisTrack` parameters as they are passed to `MovieExportToFile` (page 480). Although a movie export component can export one or more media types, it may not be able to export all the kinds of data stored in those media. The `MovieExportValidate` function allows applications to get this additional information. Movie data export components that implement this function also set the `canMovieExportValidateMovie` flag in in the `componentFlags` field of their `ComponentDescription` structure.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
ElectricImageComponent
ElectricImageComponent.win

Declared In
`QuickTimeComponents.h`

MovieImportDataRef

Undocumented

```
ComponentResult MovieImportDataRef (
   MovieImportComponent ci,
   Handle dataRef,
   OSType dataRefType,
   Movie theMovie,
   Track targetTrack,
   Track *usedTrack,
   TimeValue atTime,
   TimeValue *addedDuration,
   long inFlags,
   long *outFlags
);
```

Parameters

`ci`
A movie import component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

`dataRef`
The data reference to the data to be imported.

`dataRefType`
The type of data reference in the `dataRef` parameter.

`theMovie`
A movie identifier. Your application obtains this identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

`targetTrack`
Undocumented

usedTrack

Undocumented

atTime

Undocumented

addedDuration

Undocumented

inFlags

Flags (see below) that control the behavior of this function. See these constants:

`movieImportCreateTrack`

`movieImportInParallel`

`movieImportMustUseTrack`

`movieImportWithIdle`

outFlags

Flags (see below) that this function sets on return. See these constants:

`movieImportResultUsedMultipleTracks`

`movieImportResultNeedIdles`

`movieImportResultComplete`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

ElectricImageComponent

ElectricImageComponent.win

Declared In

`QuickTimeComponents.h`

MovieImportDoUserDialog

Requests that a component display its user dialog box.

```
ComponentResult MovieImportDoUserDialog (
   MovieImportComponent ci,
   const FSSpec *theFile,
   Handle theData,
   Boolean *canceled
);
```

Parameters

ci

A movie data import component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

theFile

A pointer to a valid file specification. If the import request pertains to a file, the application must specify the source file with this parameter and set the `theData` parameter to `NIL`. If the request is for a handle, this parameter is set to `NIL`.

theData

A handle to the data to be imported. If the import request pertains to a handle, the application must specify the source of the `data` with this parameter, and set the `theFile` parameter to `NIL`. If the request is for a file, this parameter is set to `NIL`.

canceled

A pointer to a Boolean value. Your component should set this value to TRUE if the user cancels the dialog box; otherwise, set it to FALSE.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
If your component supports a user dialog box, be sure to set the `hasMovieImportUserInterface` flag in the `componentFlags` field of your component's `ComponentDescription` structure.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Graphic Import-Export
ImportExportMovie
ImproveYourImage

Declared In
`QuickTimeComponents.h`

MovieImportDoUserDialogDataRef

Requests that a movie import component display its user dialog box.

```
ComponentResult MovieImportDoUserDialogDataRef (
   MovieImportComponent ci,
   Handle dataRef,
   OSType dataRefType,
   Boolean *canceled
);
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

dataRef

A data reference that specifies a storage location that contains the data to import.

dataRefType

The type of the data reference.

canceled

A pointer to a Boolean entity that is set to TRUE if the user cancels the export operation.

Return Value

See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error.

Discussion

This function brings up the option dialog for the import component. The data reference specified the storage location that contains the data to import.

Version Notes

Introduced in QuickTime 6.4.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`QuickTimeComponents.h`

MovieImportEstimateCompletionTime

Undocumented

```
ComponentResult MovieImportEstimateCompletionTime (
   MovieImportComponent ci,
   TimeRecord *time
);
```

Parameters

ci

A movie data import component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

time

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

MovieImportFile

Imports data from a file, using a movie data import component.

```
ComponentResult MovieImportFile (
   MovieImportComponent ci,
   const FSSpec *theFile,
   Movie theMovie,
   Track targetTrack,
   Track *usedTrack,
   TimeValue atTime,
   TimeValue *addedDuration,
   long inFlags,
   long *outFlags
);
```

Parameters

ci

A movie data import component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

theFile

A pointer to the file that contains the data that is to be imported into the `movie`. This file's type value corresponds to your component's subtype value.

theMovie

The movie for this operation. This movie identifier is supplied by the Movie Toolbox. Your component may use this identifier to add sample data to the target movie or to obtain information about the movie.

targetTrack

The track that is to receive the imported data. This track identifier is supplied by the Movie Toolbox and is valid only if the `movieImportMustUseTrack` flag in the `inFlags` parameter is set to 1.

usedTrack

A pointer to the track that received the imported data. Your component returns this track identifier to the Movie Toolbox. Your component needs to set this parameter only if you operate on a single track or if you create a new track. If you modify more than one track, leave the field referred to by this parameter unchanged.

atTime

The time corresponding to the location where your component is to place the imported data. This time value is expressed in the movie's time coordinate system.

addedDuration

A pointer to the duration of the data that your component added to the movie. Your component must specify this value in the movie's time coordinate system.

inFlags

Flags (see below) that specify control information governing the import operation. See these constants:

`movieImportCreateTrack`

`movieImportMustUseTrack`

`movieImportInParallel`

outFlags

Flags (see below) that identify a field that is to receive status information about the import operation. Your component sets the appropriate flags in this field when the operation is complete. See these constants:

`movieImportResultUsedMultipleTracks`

`movieImportInParallel`

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Your component must be prepared to perform this function at any time. You should not expect that any of your component's configuration functions will be called first. If your component can accept data from a file, be sure to set the `canMovieImportFiles` flag in the `componentFlags` field of your component's `ComponentDescription` structure.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
ConvertMovieSndTrack

Graphic Import-Export

SoundPlayer

SoundPlayer.win

SurfaceVertexProgram

Declared In
`QuickTimeComponents.h`

MovieImportGetAuxiliaryDataType

Returns the type of the auxiliary data that a component can accept.

```
ComponentResult MovieImportGetAuxiliaryDataType (
   MovieImportComponent ci,
   OSType *auxType
);
```

Parameters

`ci`
: A movie data import component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

`auxType`
: A pointer to the type of auxiliary data it can import.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function returns the type of the auxiliary data that the `ci` component can accept. For example, calling the text import component with this function indicates that the text import component will use `'styl'` information in addition to `'TEXT'` data. Note that if component includes a private component resource holding this MIME data, it can use `GetComponentResource` to retrieve it. If the resource is a public component resource, it either use `GetComponentPublicResource` with the public type and ID or `GetComponentResource` with the private type and ID.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

MovieImportGetDestinationMediaType

Returns the current type of a movie importer's destination media.

```
ComponentResult MovieImportGetDestinationMediaType (
   MovieImportComponent ci,
   OSType *mediaType
);
```

Parameters

ci
: A movie import component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

mediaType
: A pointer to a media data type; see `Data References`.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`QuickTimeComponents.h`

MovieImportGetDontBlock

Undocumented

```
ComponentResult MovieImportGetDontBlock (
   MovieImportComponent ci,
   Boolean *willBlock
);
```

Parameters

ci
: A movie data import component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

willBlock
: *Undocumented*

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

MovieImportGetFileType

Allows your movie data import component to tell the Movie Toolbox the appropriate file type for the most-recently imported movie file.

```
ComponentResult MovieImportGetFileType (
   MovieImportComponent ci,
   OSType *fileType
);
```

Parameters

ci

A movie data import component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

fileType

A pointer to an `OSType` field. Your component should place the file type value that best identifies the movie data just imported. For example, Apple's Audio CD movie data import component sets this field to `'AIFF'` whenever it creates an AIFF file instead of a movie file.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
You should implement this function only if your movie data import component creates files other than QuickTime movies. By default, the Movie Toolbox makes new files into movies, unless you override that default by providing this function.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

MovieImportGetLoadState

Undocumented

```
ComponentResult MovieImportGetLoadState (
   MovieImportComponent ci,
   long *importerLoadState
);
```

Parameters

ci

A movie data import component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

importerLoadState

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.1.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

MovieImportGetMaxLoadedTime

Undocumented

```
ComponentResult MovieImportGetMaxLoadedTime (
   MovieImportComponent ci,
   TimeValue *time
);
```

Parameters

ci

A movie data import component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

time

A pointer to a value containing the maximum loaded time.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.1.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

MovieImportGetMIMETypeList

Returns a list of MIME types supported by the movie import component.

```
ComponentResult MovieImportGetMIMETypeList (
   MovieImportComponent ci,
   QTAtomContainer *mimeInfo
);
```

Parameters

ci

A movie data import component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

mimeInfo

A pointer to a MIME type list, a QT atom container that contains a list of MIME types supported by the movie import component. The caller should dispose of the atom container when finished with it.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Your movie import component can support MIME types that correspond to formats it supports. To make a list of these MIME types available to applications or other software, it must implement this function. To indicate that your movie import component supports this function, set the `hasMovieImportMIMEList` flag in the `componentFlags` field of the `ComponentDescription` structure.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

MovieImportGetSampleDescription

Gets the current sample description for a movie import component.

```
ComponentResult MovieImportGetSampleDescription (
   MovieImportComponent ci,
   SampleDescriptionHandle *desc,
   OSType *mediaType
);
```

Parameters

ci

A movie data import component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

desc

A pointer to a handle to a `SampleDescription` structure.

mediaType

A pointer to the type of the data; see `Data References`.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

MovieImportGetSettingsAsAtomContainer

Retrieves the current settings from the movie import component.

```
ComponentResult MovieImportGetSettingsAsAtomContainer (
   MovieImportComponent ci,
   QTAtomContainer *settings
);
```

Parameters

`ci`
: A movie data import component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

`settings`
: The address where the reference to the newly created atom container should be stored by the call.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
The caller is responsible for disposing of the returned QT atom container.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Fiendishthngs

Declared In
`QuickTimeComponents.h`

MovieImportHandle

Imports data from a handle, using a movie data import component.

```
ComponentResult MovieImportHandle (
   MovieImportComponent ci,
   Handle dataH,
   Movie theMovie,
   Track targetTrack,
   Track *usedTrack,
   TimeValue atTime,
   TimeValue *addedDuration,
   long inFlags,
   long *outFlags
);
```

Parameters

ci

A movie data import component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

dataH

A handle to the data that is to be imported into the movie identified by the `theMovie` parameter. The data contained in this handle has a data type value that corresponds to your component's subtype value. Your component is not responsible for disposing of this handle.

theMovie

The movie for this operation. This movie identifier is supplied by the Movie Toolbox. Your component may use this identifier to add sample data to the target movie, or to obtain information about the movie.

targetTrack

The track that is to receive the imported data. This track identifier is supplied by the Movie Toolbox and is valid only if the `movieImportMustUseTrack` flag in the `inFlags` parameter is set to 1.

usedTrack

A pointer to the track that received the imported data. Your component returns this track identifier to the Movie Toolbox. Your component needs to set this parameter only if you operate on a single track or if you create a new track. If you modify more than one track, leave the field referred to by this parameter unchanged.

atTime

The time corresponding to the location where your component is to place the imported data. This time value is expressed in the movie's time coordinate system.

addedDuration

A pointer to the duration of the data that your component added to the movie. Your component must specify this value in the movie's time coordinate system.

inFlags

Flags (see below) that specify control information governing the import operation. See these constants:

`movieImportCreateTrack`

`movieImportMustUseTrack`

`movieImportInParallel`

outFlags

Flags (see below) that receive status information about the import operation. Your component sets the appropriate flags in this field when the operation is complete. See these constants:

`movieImportResultUsedMultipleTracks`

`movieImportInParallel`

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Your component must be prepared to perform this function at any time. You should not expect that any of your component's configuration functions will be called first. If your component can accept data from a handle, be sure to set the `canMovieImportHandles` flag in your component's `componentFlags` field.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

MovieImportIdle

Undocumented

```
ComponentResult MovieImportIdle (
   MovieImportComponent ci,
   long inFlags,
   long *outFlags
);
```

Parameters

`ci`
A movie data import component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

`inFlags`
Undocumented

`outFlags`
Undocumented

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

MovieImportSetAuxiliaryData

Provides additional data to a component.

```
ComponentResult MovieImportSetAuxiliaryData (
   MovieImportComponent ci,
   Handle data,
   OSType handleType
);
```

Parameters

ci

A movie data import component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

data

A handle to the additional data. Your component should not dispose of this handle. Be sure to copy any data you need to keep.

handleType

The type of data in the specified handle.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Your component should expect the application to call this function before the import process begins.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

MovieImportSetChunkSize

The amount of data a component works with at a time.

```
ComponentResult MovieImportSetChunkSize (
   MovieImportComponent ci,
   long chunkSize
);
```

Parameters

ci

A movie data import component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

chunkSize

The number of seconds of data your movie data import component places into each chunk of movie data. This parameter may not be set to a value less than 1.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Generally, your component should determine a reasonable default chunk size, based on the type of data you are importing. However, you may choose to allow applications to override your default value. This can be especially useful for sound data, where the chunk size affects the quality of sound playback.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

MovieImportSetDimensions

Specifies a new track's spatial dimensions.

```
ComponentResult MovieImportSetDimensions (
   MovieImportComponent ci,
   Fixed width,
   Fixed height
);
```

Parameters

`ci`

  A movie data import component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

`width`

  The width, in pixels, of the track rectangle. This parameter, along with the `height` parameter, specifies a rectangle that surrounds the image that is to be displayed when the current media is played. This value corresponds to the x coordinate of the lower-right corner of the rectangle, and it is expressed as a fixed-point number.

`height`

  The height, in pixels, of the track rectangle. This value corresponds to the y coordinate of the lower-right corner of the rectangle, and it is expressed as a fixed-point number.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

MovieImportSetDontBlock

Undocumented

```
ComponentResult MovieImportSetDontBlock (
   MovieImportComponent ci,
   Boolean dontBlock
);
```

Parameters

ci
: A movie data import component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

dontBlock
: *Undocumented*

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

MovieImportSetDuration

Controls the duration of the data that a component pastes into the target movie.

```
ComponentResult MovieImportSetDuration (
   MovieImportComponent ci,
   TimeValue duration
);
```

Parameters

ci
: A movie data import component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

duration
: The duration in the movie's time scale. If this parameter is set to 0, then you may paste any amount of movie data that is appropriate for the data to be imported.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
If your component supports paste operations (that is, your component allows the application to set the `movieImportInParallel` flag to 1 with the `MovieImportHandle` (page 493) or `MovieImportFile` (page 486) function), then you must support this function. If an application calls this function and sets a duration limit, you must abide by that limit. This function is not valid for insert operations (where the `movieImportInParallel` flag is set to 0).

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

MovieImportSetFromScrap

Indicates that the source data resides on the scrap.

```
ComponentResult MovieImportSetFromScrap (
   MovieImportComponent ci,
   Boolean fromScrap
);
```

Parameters

ci
A movie data import component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

fromScrap
Set to TRUE if the data originated on the scrap; otherwise, set to FALSE.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

MovieImportSetIdleManager

Lets a movie importer report its idling needs.

```
ComponentResult MovieImportSetIdleManager (
   MovieImportComponent ci,
   IdleManager im
);
```

Parameters

ci
A movie import component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

im
A pointer to an opaque data structure that belongs to the Mac OS Idle Manager. You get this pointer by calling `QTIdleManagerOpen` (page 273).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This routine must be implemented by a movie importer if it needs to report its idling requirements. In general, however, movie importers don't get idled. Typically, a movie importer just examines a file, scans it, and then determines if it can create a movie that will point at the file and describe how to play it. The media data is in that file, but the movie itself is in memory.

An idling importer is mostly used when you open a URL. For example, if you open an `.avi` file, the movie isn't completely constructed until the entire `.avi` file is downloaded. The job of the importer is to construct the movie, so the importer isn't going to be done constructing the movie until it is downloaded, which means you can't fast start an AVI movie. So the AVI importer returns immediately with a movie that is partially constructed. Every time QuickTime gets tasked, it gets some more time, but you can go ahead and start playing because it has already returned a movie, though one that is not complete yet.

An idling importer can return even before there's enough downloaded to construct a movie. It just creates an empty movie with no tracks and keep idling it, and eventually a movie appears.

Version Notes
Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`QuickTimeComponents.h`

MovieImportSetMediaDataRef

Specifies a storage location that is to receive imported movie data.

```
ComponentResult MovieImportSetMediaDataRef (
   MovieImportComponent ci,
   Handle dataRef,
   OSType dataRefType
);
```

Parameters

`ci`
The component instance that identifies your connection to the graphics importer component.

`dataRef`
A data reference that specifies a storage location that receives the imported data.

`dataRefType`
The type of the data reference.

Return Value
See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error.

Discussion
By calling this function you can specify a storage location that receives some imported movie data.

Version Notes
Introduced in QuickTime 6.4.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`QuickTimeComponents.h`

MovieImportSetMediaFile

Specifies a media file that is to receive the imported movie data.

```
ComponentResult MovieImportSetMediaFile (
   MovieImportComponent ci,
   AliasHandle alias
);
```

Parameters

ci
: A movie data import component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

alias
: The media file that is to receive the imported movie data. Your component must make a copy of this parameter. You should not dispose of it.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

MovieImportSetNewMovieFlags

Implemented by a movie import component to determine the original flags for NewMovieFromDataRef.

```
ComponentResult MovieImportSetNewMovieFlags (
   MovieImportComponent ci,
   long newMovieFlags
);
```

Parameters

ci
: A movie import component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

newMovieFlags
: Constants (see below) that control characteristics of the new movie. See these constants:
  - `newMovieActive`
  - `newMovieDontResolveDataRefs`
  - `newMovieDontAskUnresolvedDataRefs`

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`QuickTimeComponents.h`

MovieImportSetOffsetAndLimit

Specifies location and size of data that should be imported.

```
ComponentResult MovieImportSetOffsetAndLimit (
   MovieImportComponent ci,
   unsigned long offset,
   unsigned long limit
);
```

Parameters

ci
: A movie data import component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

offset
: A byte offset into the file that indicates where the import operation begins.

limit
: A byte offset into the file that indicates the last data in the file that can be imported.

Return Value
See `Error Codes`. Returns `badComponentSelector` if the movie import component does not support this function. Returns `noErr` if there is no error.

Discussion
Typically, this function is used when the data is from a part of a file rather than the entire file. It is especially useful when one file format is embedded in another; it allows your application to skip header data for the enclosing file and begin importing data at the start of the desired format.

Special Considerations

Not all movie import components support this function. Those that do include the movie import components for the `kQTFileTypeAIFF`, `kQTFileTypeWave`, and `kQTFileTypeMuLaw` file types. Those that do not return the `badComponentSelector` result code in response to a this call.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

MovieImportSetOffsetAndLimit64

Specifies location and size of data that should be imported from a file.

```
ComponentResult MovieImportSetOffsetAndLimit64 (
   MovieImportComponent ci,
   const wide *offset,
   const wide *limit
);
```

Parameters

ci

A movie data import component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

offset

A byte offset into the file that indicates where the import operation begins.

limit

A byte offset into the file that indicates the last data in the file that can be imported.

Return Value

See `Error Codes`. Returns `badComponentSelector` if the movie import component does not support this function. Returns `noErr` if there is no error.

Discussion

This function serves the same purpose as `MovieImportSetOffsetAndLimit` (page 502). The only difference is that the offset and limit can hold 64-bit offsets. This function is especially useful when one file format is embedded in another; it allows your application to skip header data for the enclosing file and begin importing data at the start of the desired format.

Special Considerations

Not all movie import components support this function. Those that do not return the `badComponentSelector` result code. If this function is not implemented and the offset and limit can be expressed using 32-bit offsets, `MovieImportSetOffsetAndLimit` (page 502) should be tried.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

MovieImportSetProgressProc

Assigns a movie progress function.

```
ComponentResult MovieImportSetProgressProc (
   MovieImportComponent ci,
   MovieProgressUPP proc,
   long refcon
);
```

Parameters

ci

A movie data import component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

proc

A pointer to the application's `MovieProgressProc` callback. If this parameter is set to `NIL`, the application is removing its progress function. In this case, your component should stop calling the progress function.

refcon

Specifies a reference constant. Your component should pass this constant back to the application's progress function whenever you call that function. The application may use this parameter to point to a data structure containing any information the callback needs.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
The `MovieProgressProc` callback interface not only allows you to report progress to the application, but also allows the application to cancel the request.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

MovieImportSetSampleDescription

Provides a SampleDescription structure to a movie data import component.

```
ComponentResult MovieImportSetSampleDescription (
   MovieImportComponent ci,
   SampleDescriptionHandle desc,
   OSType mediaType
);
```

Parameters

ci

A movie data import component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

desc

A handle to a `SampleDescription` structure. Your component must not dispose of this handle. If you want to save any data from the structure, be sure to copy it at this time.

mediaType

The type of sample description referred to by the `desc` parameter. If the `desc` parameter refers to an `ImageDescription` structure, this parameter is set to `VideoMediaType` (`'vide'`); for `SoundDescription` structures, this parameter is set to `SoundMediaType` (`'soun'`).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

MovieImportSetSampleDuration

Sets the sample duration for new samples to be created with a component.

```
ComponentResult MovieImportSetSampleDuration (
   MovieImportComponent ci,
   TimeValue duration,
   TimeScale scale
);
```

Parameters

ci

A movie data import component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

duration

The sample duration in units specified by the `scale` parameter.

scale

The time scale for the duration value. This may be any arbitrary time scale; that is, it may not correspond to the movie's time scale. You should convert this time scale to the movie's time scale before using the duration value, using `ConvertTimeScale` (page 183).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

MovieImportSetSettingsFromAtomContainer

Sets the movie import component's current configuration from the passed settings data.

```
ComponentResult MovieImportSetSettingsFromAtomContainer (
   MovieImportComponent ci,
   QTAtomContainer settings
);
```

Parameters

ci

A movie data import component instance. Your software obtains this reference from OpenComponent or OpenDefaultComponent.

settings

A QT atom container containing settings.

Return Value

See Error Codes. Returns noErr if there is no error.

Discussion

The settings QT atom container may contain atoms other than those expected by the particular component type or may be missing certain atoms. The function uses only those settings it understands.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

QuickTimeComponents.h

MovieImportValidate

Allows your movie data import component to validate the data to be passed to your component.

```
ComponentResult MovieImportValidate (
   MovieImportComponent ci,
   const FSSpec *theFile,
   Handle theData,
   Boolean *valid
);
```

Parameters

ci

A movie data import component instance. Your software obtains this reference from OpenComponent or OpenDefaultComponent.

theFile

An FSSpec structure that defines the file to validate if the importer imports from files.

theData

The data to validate if the importer imports from handles.

valid

A pointer to a Boolean value. If the data or file can be imported, the value returned is TRUE. Otherwise, it returns FALSE.

Return Value

See Error Codes. Returns noErr if there is no error.

Discussion
Movie import components can implement this function to allow applications to determine if a given file or handle to data is acceptable for a particular import component. As this function may be called on many files, the validation process should be as fast as possible.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

MovieImportValidateDataRef

Validates the data file indicated by the data reference.

```
ComponentResult MovieImportValidateDataRef (
   MovieImportComponent ci,
   Handle dataRef,
   OSType dataRefType,
   UInt8 *valid
);
```

Parameters

ci
: A movie data import component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

dataRef
: The data reference to the file to be validated.

dataRefType
: The type of data reference for the `dataRef` parameter.

valid
: A pointer to a UInt8 value. If the data or file cannot be imported, the value returned should be 0. Otherwise, it should be set to 128.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Movie import components can implement this function to allow applications to determine if a given file referenced by a data reference is acceptable for a particular import component. The data reference can refer to any data for which there is a suitable data handler component installed and available to QuickTime. As this function may be called on many files, the validation process should be as fast as possible. Furthermore, the importer should probably limit the amount of reading it performs, especially when the data handler refers to data on the Internet.

Special Considerations

Unlike `MovieImportValidate` (page 506), the `valid` parameter for this function is a value that can be interpreted as the degree to which the importer can interpret the file's contents. In all cases, returning 0 indicates the file cannot be interpreted by the importer. However, other non-zero values can be used to determine the relative weighting between multiple importers that can import a particular kind of file. For now, it is best to return either 0 or 128 only.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

ElectricImageComponent

ElectricImageComponent.win

Declared In

`QuickTimeComponents.h`

NewMovieExportGetDataUPP

Allocates a Universal Procedure Pointer for the MovieExportGetDataProc callback.

```
MovieExportGetDataUPP NewMovieExportGetDataUPP (
   MovieExportGetDataProcPtr userRoutine
);
```

Parameters

`userRoutine`

A pointer to your application-defined function.

Return Value

A new UPP; see `Universal Procedure Pointers`.

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes

Introduced in QuickTime 4.1. Replaces `NewMovieExportGetDataProc`.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CIVideoDemoGL

qtmoviefromprocs

qtmoviefromprocs.win

Declared In

`QuickTimeComponents.h`

NewMovieExportGetPropertyUPP

Allocates a Universal Procedure Pointer for the MovieExportGetPropertyProc callback.

```
MovieExportGetPropertyUPP NewMovieExportGetPropertyUPP (
   MovieExportGetPropertyProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your application-defined function.

Return Value

A new UPP; see `Universal Procedure Pointers`.

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes

Introduced in QuickTime 4.1. Replaces `NewMovieExportGetPropertyProc`.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CIVideoDemoGL

qtmoviefromprocs

qtmoviefromprocs.win

Declared In

`QuickTimeComponents.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`.

Return Value

A new Universal Procedure Pointer that you will use to invoke your callback.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`QuickTimeComponents.h`

NewSCModalFilterUPP

Allocates a Universal Procedure Pointer for the SCModalFilterProc callback.

```
SCModalFilterUPP NewSCModalFilterUPP (
   SCModalFilterProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your application-defined function.

Return Value

A new UPP; see `Universal Procedure Pointers`.

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes

Introduced in QuickTime 4.1. Replaces `NewSCModalFilterProc`.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qtcompress

qtcompress.win

Declared In

`QuickTimeComponents.h`

NewSCModalHookUPP

Allocates a Universal Procedure Pointer for the SCModalHookProc callback.

```
SCModalHookUPP NewSCModalHookUPP (
   SCModalHookProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your application-defined function.

Return Value

A new UPP; see `Universal Procedure Pointers`.

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes

Introduced in QuickTime 4.1. Replaces `NewSCModalHookProc`.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

ConvertToMovieJr

qtcompress

qtcompress.win

VideoProcessing

Declared In

`QuickTimeComponents.h`

PreviewEvent

May be called as appropriate if a preview component handles events.

```
ComponentResult PreviewEvent (
   pnotComponent p,
   EventRecord *e,
   Boolean *handledEvent
);
```

Parameters

p

Specifies your preview component. You obtain this identifier from `OpenComponent`.

e

A pointer to the event structure for this operation.

handledEvent

A pointer to a Boolean value. If you completely handle an event such as a mouse-down event or keystroke, you should set the `handledEvent` parameter to TRUE. Otherwise, set it to FALSE.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

PreviewMakePreview

Creates previews by allocating a handle to data that is to be added to a file.

```
ComponentResult PreviewMakePreview (
   pnotComponent p,
   OSType *previewType,
   Handle *previewResult,
   const FSSpec *sourceFile,
   ICMProgressProcRecordPtr progress
);
```

Parameters

p

Specifies your preview component. You obtain this identifier from `OpenComponent`.

previewType

A pointer to the type of preview component that should be used to display the preview.

previewResult

A pointer to a handle of cached preview data created by this function.

sourceFile

A pointer to a reference to the file for which the preview is created.

progress

A pointer to an `ICMProgressProcRecord` structure.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

PreviewMakePreviewReference

Returns the type and identification number of a resource within a file to be used as the preview for a file.

```
ComponentResult PreviewMakePreviewReference (
   pnotComponent p,
   OSType *previewType,
   short *resID,
   const FSSpec *sourceFile
);
```

Parameters

p

Specifies your preview component. You obtain this identifier from `OpenComponent`.

previewType

A pointer to the type of preview component that should be used to display the preview.

resID

A pointer to the identification number of a resource within the file to be used as the preview for the file.

sourceFile

A pointer to an `FSSpec` structure that provides a reference to the file for which the preview is created.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

PreviewShowData

Displays a preview if it does not handle events.

```
ComponentResult PreviewShowData (
   pnotComponent p,
   OSType dataType,
   Handle data,
   const Rect *inHere
);
```

Parameters

p

Specifies your preview component. You obtain this identifier from `OpenComponent`.

dataType

The type of handle pointing to the data to be displayed in the preview.

data

A handle to the data, which is typically the same as the subtype of your preview component.

inHere

A pointer to a `Rect` structure that defines the area into which you draw the preview. The current port is set to the correct graphics port for drawing. You must not draw outside the given rectangle.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
AlwaysPreview

Declared In
`QuickTimeComponents.h`

SCAsyncIdle

Called occasionally while performing asynchronous compression with SCCompressSequenceFrameAsync.

```
ComponentResult SCAsyncIdle (
   ComponentInstance ci
);
```

Parameters

ci

Your application's connection to the image-compression component being used by `SCCompressSequenceFrameAsync` (page 520). You obtain this identifier from `OpenDefaultComponent`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qtcompress

qtcompress.win

Declared In

`QuickTimeComponents.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.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`QuickTimeComponents.h`

SCCompressImage

Compresses an image that is stored in a PixMap structure.

```
ComponentResult SCCompressImage (
   ComponentInstance ci,
   PixMapHandle src,
   const Rect *srcRect,
   ImageDescriptionHandle *desc,
   Handle *data
);
```

Parameters

ci

Identifies your application's connection to a standard image-compression dialog component. You obtain this identifier from `OpenDefaultComponent`.

src

A handle to the `PixMap` structure to be compressed.

srcRect

A pointer to a portion of the `PixMap` structure to compress as a `Rect` structure. This rectangle must be in the pixel map's coordinate system. If you want to compress the entire pixel map, set this parameter to `NIL`.

desc

A pointer to a handle to an `ImageDescription` structure. The standard dialog component creates an `ImageDescription` structure when it compresses the image, and returns a handle to that structure in the field referred to by this parameter. The component sizes that handle appropriately. Your application is responsible for disposing of that handle when you are done with it.

data

A pointer to a handle. The standard dialog component returns a handle to the compressed image data in the field referred to by this parameter. The component sizes that handle appropriately. Your application is responsible for disposing of that handle when you are done with it.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qtcompress

qtcompress.win

qtstdcompr

qtstdcompr.win

Declared In

`QuickTimeComponents.h`

SCCompressPicture

Compresses a Picture structure that is stored by a handle.

```
ComponentResult SCCompressPicture (
   ComponentInstance ci,
   PicHandle srcPicture,
   PicHandle dstPicture
);
```

Parameters

ci

Identifies your application's connection to a standard image-compression dialog component. You obtain this identifier from `OpenDefaultComponent`.

srcPicture

A handle to the `Picture` structure to be compressed.

dstPicture

A handle to the compressed `Picture` structure. The standard dialog component resizes this handle to accommodate the compressed structure. Your application is responsible for creating and disposing of this handle when you are done with it.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SCCompressPictureFile

Compresses a Picture structure that is stored in a file.

```
ComponentResult SCCompressPictureFile (
   ComponentInstance ci,
   short srcRefNum,
   short dstRefNum
);
```

Parameters

ci

Identifies your application's connection to a standard image-compression dialog component. You obtain this identifier from `OpenDefaultComponent`.

srcRefNum

A reference to the file to be compressed.

dstRefNum

A reference to the file that is to receive the compressed data. This may be the same as the source file. The standard dialog component places the compressed image data into the file identified by this reference. Your application is responsible for this file after the compression operation.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Std Compression Examples

Declared In
`QuickTimeComponents.h`

SCCompressSequenceBegin

Initiates a sequence-compression operation.

```
ComponentResult SCCompressSequenceBegin (
   ComponentInstance ci,
   PixMapHandle src,
   const Rect *srcRect,
   ImageDescriptionHandle *desc
);
```

Parameters

ci

Identifies your application's connection to a standard image-compression component. You obtain this identifier from `OpenDefaultComponent`.

src

A handle to the `PixMap` structure to be compressed. This pixel map must contain the first image in the sequence.

srcRect

A pointer to a portion of the `PixMap` structure to compress as a `Rect` structure. This rectangle must be in the pixel map's coordinate system. If you want to compress the entire structure, set this parameter to `NIL`.

desc

A pointer to an image description handle. The standard dialog component creates an image description structure when it compresses the image, and returns a handle to that structure in the field referred to by this parameter. The component sizes the handle appropriately. If you do not want this information, set this parameter to `NIL`.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CompressMovies

ConvertToMovieJr

qtcompress

qtcompress.win

VideoProcessing

Declared In
`QuickTimeComponents.h`

SCCompressSequenceEnd

Ends a sequence-compression operation.

```
ComponentResult SCCompressSequenceEnd (
   ComponentInstance ci
);
```

Parameters

ci

Identifies your application's connection to a standard image-compression component. You obtain this identifier from `OpenDefaultComponent`.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
The standard dialog component disposes of any memory it used to compress the image sequence, including the data and image description buffers. You must call this function once for each sequence you start.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CompressMovies

ConvertToMovieJr

qtcompress

qtcompress.win

VideoProcessing

Declared In
`QuickTimeComponents.h`

SCCompressSequenceFrame

Continues a sequence-compression operation.

```
ComponentResult SCCompressSequenceFrame (
   ComponentInstance ci,
   PixMapHandle src,
   const Rect *srcRect,
   Handle *data,
   long *dataSize,
   short *notSyncFlag
);
```

Parameters

ci
: Identifies your application's connection to a standard image-compression component. You obtain this identifier from `OpenDefaultComponent`.

src
: A handle to the `PixMap` structure to be compressed.

srcRect
: A pointer to a portion of the `PixMap` structure to compress as a `Rect` structure. This rectangle must be in the pixel map's coordinate system. If you want to compress the entire pixel map, set this parameter to `NIL`.

data
: A pointer to a handle. The standard compression component returns a handle to the compressed image data in the field referred to by this parameter. The component sizes that handle appropriately for the sequence.

dataSize
: A pointer to a long integer. The standard compression component returns a value that indicates the number of bytes of compressed image data that it returns. Note that this value will differ from the size of the handle referred to by the `data` parameter, because the handle is allocated to accommodate the largest image in the sequence.

notSyncFlag
: A pointer to a short integer that indicates whether the compressed frame is a key frame. If the frame is a key frame, the standard compression component sets the field referred to by this parameter to 0; otherwise, the `component` sets this field to `mediaSampleNotSync`. You may use this field to set the `sampleFlags` parameter of the `AddMediaSample` (page 1536) function.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
You must call this function once for each frame in the sequence, including the first frame.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CompressMovies

ConvertToMovieJr

qtcompress

qtcompress.win

VideoProcessing

Declared In

`QuickTimeComponents.h`

SCCompressSequenceFrameAsync

An asynchronous variant of SCCompressSequenceFrame, with a completion callback.

```
ComponentResult SCCompressSequenceFrameAsync (
   ComponentInstance ci,
   PixMapHandle src,
   const Rect *srcRect,
   Handle *data,
   long *dataSize,
   short *notSyncFlag,
   ICMCompletionProcRecordPtr asyncCompletionProc
);
```

Parameters

ci

Identifies your application's connection to a standard image-compression component. You obtain this identifier from `OpenDefaultComponent`.

src

A handle to the `PixMap` structure to be compressed.

srcRect

A pointer to a portion of the `PixMap` structure to compress as a `Rect` structure. This rectangle must be in the pixel map's coordinate system. If you want to compress the entire pixel map, set this parameter to `NIL`.

data

A pointer to a handle. The standard compression component returns a handle to the compressed image data in the field referred to by this parameter. The component sizes that handle appropriately for the sequence.

dataSize

A pointer to a long integer. The standard compression component returns a value that indicates the number of bytes of compressed image data that it returns. Note that this value will differ from the size of the handle referred to by the `data` parameter, because the handle is allocated to accommodate the largest image in the sequence.

notSyncFlag

A pointer to a short integer that indicates whether the compressed frame is a key frame. If the frame is a key frame, the standard compression component sets the field referred to by this parameter to 0; otherwise, the `component` sets this field to `mediaSampleNotSync`. You may use this field to set the `sampleFlags` parameter of the `AddMediaSample` (page 1536) function.

asyncCompletionProc

A pointer to an `ICMCompletionProcRecord` structure. If you pass `NIL`, the `SCCompressSequenceFrameAsync` function acts like `SCCompressSequenceFrame` (page 518).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

While performing asynchronous compression with this function, you should occasionally call `SCAsyncIdle` (page 514). This gives the standard compression component an opportunity to restart its compression operation if it needs to force a key frame.

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qtcompress

qtcompress.win

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` (page 35)().

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

OpenGLCaptureToMovie

Quartz Composer QCTV

Declared In

`QuickTimeComponents.h`

SCDefaultPictFileSettings

Derives default compression settings for a Picture structure that is stored in a file.

```
ComponentResult SCDefaultPictFileSettings (
   ComponentInstance ci,
   short srcRef,
   short motion
);
```

Parameters

ci

Identifies your application's connection to a standard image-compression dialog component. You obtain this identifier from `OpenDefaultComponent`.

srcRef

A reference to the file to be analyzed.

motion

Specifies whether the image is part of a sequence. Set this parameter to TRUE if the image is part of a sequence; set it to FALSE if you are working with a single still image.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Std Compression Examples

Declared In

`QuickTimeComponents.h`

SCDefaultPictHandleSettings

Derives default compression settings for a Picture structure that is stored by a handle.

```
ComponentResult SCDefaultPictHandleSettings (
   ComponentInstance ci,
   PicHandle srcPicture,
   short motion
);
```

Parameters

ci

Identifies your application's connection to a standard image-compression dialog component. You obtain this identifier from `OpenDefaultComponent`.

srcPicture

A handle to the `Picture` structure to be analyzed.

motion

Specifies whether the image is part of a sequence. Set this parameter to TRUE if the image is part of a sequence; set it to FALSE if you are working with a single still image.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SCDefaultPixMapSettings

Derives default compression settings for an image that is stored in a pixel map.

```
ComponentResult SCDefaultPixMapSettings (
   ComponentInstance ci,
   PixMapHandle src,
   short motion
);
```

Parameters

ci

Identifies your application's connection to a standard image-compression dialog component. You obtain this identifier from `OpenDefaultComponent`.

src

A handle to the `PixMap` structure to be analyzed.

motion

Specifies whether the image is part of a sequence. Set this parameter to TRUE if the image is part of a sequence; set it to FALSE if you are working with a single still image.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CompressMovies

ConvertToMovieJr

qtcompress

qtcompress.win

VideoProcessing

Declared In

`QuickTimeComponents.h`

SCGetBestDeviceRect

Determines the boundary rectangle that surrounds the display device that supports the largest color or grayscale palette.

```
ComponentResult SCGetBestDeviceRect (
   ComponentInstance ci,
   Rect *r
);
```

Parameters

ci

Identifies your application's connection to a standard image-compression dialog component. You obtain this identifier from `OpenDefaultComponent`.

r

A pointer to a `Rect` structure. The function returns the global coordinates of a rectangle that surrounds the appropriate display device.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The standard image-compression dialog component uses this function to position rectangles and dialog boxes when you indicate that the component is to choose the best display device. It subtracts the menu bar from the returned rectangle if the best device is also the main display device.

Special Considerations

In general, your application does not need to use this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SCGetCompressFlags

Gets compression flags for a standard image-compression dialog component.

```
ComponentResult SCGetCompressFlags (
   ComponentInstance ci,
   long *flags
);
```

Parameters

ci

Identifies your application's connection to a standard image-compression dialog component. You obtain this identifier from `OpenDefaultComponent`.

flags

A pointer to compression flags (see below). See these constants:

`scCompressFlagIgnoreIdenticalFrames`

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SCGetCompressionExtended

Undocumented

```
ComponentResult SCGetCompressionExtended (
   ComponentInstance ci,
   SCParams *params,
   Point where,
   SCModalFilterUPP filterProc,
   SCModalHookUPP hookProc,
   long refcon,
   StringPtr customName
);
```

Parameters

ci
: Identifies your application's connection to a standard image-compression dialog component. You obtain this identifier from `OpenDefaultComponent`.

params
: A pointer to an `SCParams` structure.

where
: *Undocumented*

filterProc
: A Universal Procedure Pointer that accesses a `SCModalFilterProc` callback.

hookProc
: A Universal Procedure Pointer that accesses a `SCModalHookProc` callback.

refcon
: A reference constant to be passed to your callbacks. Use this parameter to point to a data structure containing any information your callbacks need.

customName
: *Undocumented*

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SCGetInfo

Retrieves configuration information from the standard dialog component.

```
ComponentResult SCGetInfo (
   ComponentInstance ci,
   OSType infoType,
   void *info
);
```

Parameters

ci

Identifies your application's connection to a standard image-compression dialog component. You obtain this identifier from `OpenDefaultComponent`.

infoType

A constant (see below) that specifies the type of information you want to retrieve. See these constants:

`scSpatialSettingsType`

`scTemporalSettingsType`

`scDataRateSettingsType`

`scColorTableType`

`scProgressProcType`

`scExtendedProcsType`

`scPreferenceFlagsType`

`scSettingsStateType`

`scSequenceIDType`

`scWindowPositionType`

`scCodecFlagsType`

info

A pointer to a field that is to receive the information. The `infoType` constant descriptions (see below) include information about this field.

Return Value
See `Error Codes`. If the component cannot satisfy your request, it returns a result code of `scTypeNotFoundErr`. It returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CompressMovies

ConvertMovieSndTrack

ConvertToMovieJr

qtcompress

VideoProcessing

Declared In
`QuickTimeComponents.h`

SCGetSettingsAsAtomContainer

Places the current configuration from the standard image-compression component in a QT atom container.

```
ComponentResult SCGetSettingsAsAtomContainer (
   ComponentInstance ci,
   QTAtomContainer *settings
);
```

Parameters

ci

The standard compression component instance.

settings

The address where the newly-created atom container should be stored.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
The caller is responsible for disposing of the returned QT atom container.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
ConvertMovieSndTrack

ElectricImageComponent.win

Quartz Composer QCTV

ThreadsExportMovie

vrmakepano

Declared In
`QuickTimeComponents.h`

SCGetSettingsAsText

Undocumented

```
ComponentResult SCGetSettingsAsText (
   ComponentInstance ci,
   Handle *text
);
```

Parameters

ci

Identifies your application's connection to a standard image-compression dialog component. You obtain this identifier from `OpenDefaultComponent`.

text

A pointer to a handle to text.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SCNewGWorld

Creates a graphics world based on the current compression settings.

```
ComponentResult SCNewGWorld (
   ComponentInstance ci,
   GWorldPtr *gwp,
   Rect *rp,
   GWorldFlags flags
);
```

Parameters

ci

Identifies your application's connection to a standard image-compression dialog component. You obtain this identifier from `OpenDefaultComponent`.

gwp

A pointer to a pointer to a `CGrafPort` structure that defines a graphics world. The standard dialog component places a pointer to the new graphics world into the field referred to by this parameter. If the component cannot create the graphics world, it sets this field to `NIL`.

rp

A pointer to the boundaries of the graphics world. If you set this parameter to `NIL`, the standard dialog component uses the test image's boundary rectangle. If you don't specify a boundary rectangle and there is no test image, the component does not create the graphics world.

flags

Contains flags (see below) that determine some of the memory characteristics of the new graphics world. See these constants:

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SCPositionDialog

Helps position a dialog box on the screen.

```
ComponentResult SCPositionDialog (
   ComponentInstance ci,
   short id,
   Point *where
);
```

Parameters

ci

Identifies your application's connection to a standard image-compression dialog component. You obtain this identifier from `OpenDefaultComponent`.

id

The resource number of a `'DLOG'` resource. The function positions the dialog box that corresponds to this resource.

where

A pointer to a `Point` structure identifying the desired location of the upper-left corner of the dialog box in global coordinates. This parameter allows you to indicate how you want to position the dialog box on the screen.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

ConvertToMovieJr

VideoProcessing

Declared In

`QuickTimeComponents.h`

SCPositionRect

Positions a rectangle on the screen.

```
ComponentResult SCPositionRect (
   ComponentInstance ci,
   Rect *rp,
   Point *where
);
```

Parameters

ci

Identifies your application's connection to a standard image-compression dialog component. You obtain this identifier from `OpenDefaultComponent`.

rp

A pointer to a `Rect` structure. When you call the function, this structure should contain the rectangle's current global coordinates. The function adjusts the coordinates in the structure to reflect the rectangle's new position.

where

A pointer to a `Point` structure identifying the desired location of the upper-left corner of the rectangle in global coordinates. This parameter allows your application to position the rectangle on the screen.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CompressMovies

ConvertToMovieJr

VideoProcessing

Declared In

`QuickTimeComponents.h`

SCRequestImageSettings

Displays the standard image dialog box to the user and shows default settings you have established.

```
ComponentResult SCRequestImageSettings (
   ComponentInstance ci
);
```

Parameters

ci

Identifies your application's connection to a standard image-compression dialog component. You obtain this identifier from `OpenDefaultComponent`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Use this function to retrieve the user's preferences for compressing a single image; use `SCRequestSequenceSettings` (page 531) when you are working with an image sequence. Both functions manipulate the compression settings that the component stores for you.

The component derives the current settings when you may supply an image to the component from which it can derive default settings. If you have not set any defaults, but you do supply a test image for the dialog, the component examines the test image and derives appropriate default values based upon its characteristics. If you have not set any defaults and do not supply a test image, the component uses its own default values.

Special Considerations

You may modify the settings by using `SCSetInfo` (page 532). You may customize the dialog boxes by specifying a modal-dialog hook function or a custom button. You may use the custom button to invoke an ancillary dialog box that is specific to your application.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
BurntTextSampleCode

qtstdcompr.win

SCAudioCompress

Std Compression Examples

WhackedTV

Declared In
`QuickTimeComponents.h`

SCRequestSequenceSettings

Displays the standard sequence dialog box to the user and shows default settings you have established.

```
ComponentResult SCRequestSequenceSettings (
   ComponentInstance ci
);
```

Parameters

`ci`

Identifies your application's connection to a standard image-compression dialog component. You obtain this identifier from `OpenDefaultComponent`.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Use `SCRequestSequenceSettings` to retrieve the user's preferences for compressing an image sequence; use `SCRequestImageSettings` (page 530) when you are working with a single image. Both functions manipulate the compression settings that the component stores for you.

The component derives the current settings when you may supply an image to the component from which it can derive default settings. If you have not set any defaults, but you do supply a test image for the dialog, the component examines the test image and derives appropriate default values based upon its characteristics. If you have not set any defaults and do not supply a test image, the component uses its own default values.

Special Considerations

You may modify the settings by using `SCSetInfo` (page 532). You may customize the dialog boxes by specifying a modal-dialog hook function or a custom button. You may use the custom button to invoke an ancillary dialog box that is specific to your application.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
ConvertToMovieJr

OpenGLCaptureToMovie

qtcompress

qtcompress.win

Quartz Composer QCTV

Declared In
`QuickTimeComponents.h`

SCSetCompressFlags

Sets compression flags for a standard image-compression dialog component.

```
ComponentResult SCSetCompressFlags (
   ComponentInstance ci,
   long flags
);
```

Parameters

`ci`

Identifies your application's connection to a standard image-compression dialog component. You obtain this identifier from `OpenDefaultComponent`.

`flags`

Flags (see below) to set. See these constants:

`scCompressFlagIgnoreIdenticalFrames`

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SCSetInfo

Modifies the standard dialog component's configuration information.

```
ComponentResult SCSetInfo (
   ComponentInstance ci,
   OSType infoType,
   void *info
);
```

Parameters

ci

Identifies your application's connection to a standard image-compression dialog component. You obtain this identifier from `OpenDefaultComponent`.

infoType

A constant (see below) that specifies the type of information you want to set. See these constants:

`scSpatialSettingsType`

`scTemporalSettingsType`

`scDataRateSettingsType`

`scColorTableType`

`scProgressProcType`

`scExtendedProcsType`

`scPreferenceFlagsType`

`scSettingsStateType`

`scSequenceIDType`

`scWindowPositionType`

`scCodecFlagsType`

info

A pointer to a field that contains the new information. The `infoType` constant descriptions (see below) include information about this field.

Return Value

See `Error Codes`. If the component cannot satisfy your request, it returns a result code of `scTypeNotFoundErr`. It returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CompressMovies

ConvertToMovieJr

qtcompress

qtcompress.win

VideoProcessing

Declared In

`QuickTimeComponents.h`

SCSetSettingsFromAtomContainer

Sets the standard image-compression component's current configuration from data in a QT atom container.

```
ComponentResult SCSetSettingsFromAtomContainer (
   ComponentInstance ci,
   QTAtomContainer settings
);
```

Parameters

ci

Standard compression component instance.

settings

A QT atom container reference to the settings.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The settings QT atom container may contain atoms other than those expected by the particular component type or may be missing certain atoms. The function will only use settings it understands.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

ConvertMovieSndTrack

OpenGLCaptureToMovie

Quartz Composer QCTV

Declared In

`QuickTimeComponents.h`

SCSetTestImagePictFile

Sets the dialog box's test image from a Picture structure that is stored in a picture file.

```
ComponentResult SCSetTestImagePictFile (
   ComponentInstance ci,
   short testFileRef,
   Rect *testRect,
   short testFlags
);
```

Parameters

ci

Identifies your application's connection to a standard image-compression dialog component. You obtain this identifier from `OpenDefaultComponent`.

testFileRef
Identifies the file that contains the new test image. Your application is responsible for opening this file before calling this function. You must also close the file when you are done with it. You must clear the image or close your connection to the standard image-compression dialog component before you close the file. If the file contains a large image, the component may take some time to display the standard image-compression dialog box. In this case, the component displays the watch cursor while it loads the test image.

testRect
A pointer to a `Rect` structure. This rectangle specifies, in the coordinate system of the source image, the area of interest or point of interest in the test image. The area of interest defines a portion of the test image that is to be shown to the user in the dialog box. Use this parameter to direct the component to a specific portion of the test image. The component uses the value of the `testFlags` parameter to determine how it transforms large images before displaying them to the user.

testFlags
Constants (see below) that specify how the component is to display a test image that is larger than the test image portion of the dialog box. If you set this parameter to 0, the component uses a default method of its own choosing. In all cases, the component centers the area or point of interest in the test image portion of the dialog box, and then displays some part of the test image. See these constants:

`scPreferCropping`

`scPreferScaling`

`scPreferScalingAndCropping`

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Std Compression Examples

Declared In
`QuickTimeComponents.h`

SCSetTestImagePictHandle

Sets the dialog box's test image from a Picture structure that is stored in a handle.

```
ComponentResult SCSetTestImagePictHandle (
   ComponentInstance ci,
   PicHandle testPict,
   Rect *testRect,
   short testFlags
);
```

Parameters

ci

Identifies your application's connection to a standard image-compression dialog component. You obtain this identifier from `OpenDefaultComponent`.

testPict

Identifies a handle that contains the new test image. Your application is responsible for disposing of this handle when you are done with it. You must clear the image or close your connection to the standard image-compression dialog component before you dispose of this handle or close the corresponding resource file. You must set this handle as nonpurgeable.

testRect

A pointer to a `Rect` structure. This structure specifies, in the coordinate system of the source image, the area of interest or point of interest in the test image. The area of interest defines a portion of the test image that is to be shown to the user in the dialog box. Use this parameter to direct the component to a specific portion of the test image. The component uses the value of the `testFlags` parameter to determine how it transforms this image before displaying it to the user. The component uses the `testFlags` parameter only when the test image is larger than the test image portion of the dialog box.

testFlags

Constants (see below) that specify how the component is to display a test image that is larger than the test image portion of the dialog box. If you set this parameter to 0, the component uses a default method of its own choosing. In all cases, the component centers the area or point of interest in the test image portion of the dialog box, and then displays some part of the test image. See these constants:

`scPreferCropping`

`scPreferScaling`

`scPreferScalingAndCropping`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SCSetTestImagePixMap

Sets the dialog box's test image from a Picture structure that is stored in a PixMap structure.

```
ComponentResult SCSetTestImagePixMap (
   ComponentInstance ci,
   PixMapHandle testPixMap,
   Rect *testRect,
   short testFlags
);
```

Parameters

ci

Identifies your application's connection to a standard image-compression dialog component. You obtain this identifier from `OpenDefaultComponent`.

testPixMap

A handle to a `PixMap` structure that contains the new test image. Your application is responsible for creating this structure before calling the function. You must also dispose of the structure when you are done with it. You must clear the image or close your connection to the standard image-compression dialog component before you dispose of the structure.

testRect

A pointer to a `Rect` structure. This rectangle specifies, in the coordinate system of the source image, the area of interest or point of interest in the test image. The area of interest defines a portion of the test image that is to be shown to the user in the dialog box. Use this parameter to direct the component to a specific portion of the test image. The component uses the value of the `testFlags` parameter to determine how it transforms large images before displaying them to the user.

testFlags

Constants (see below) that specify how the component is to display a test image that is larger than the test image portion of the dialog box. If you set this parameter to 0, the component uses a default method of its own choosing. In all cases, the component centers the area or point of interest in the test image portion of the dialog box, and then displays some part of the test image. See these constants:

`scPreferCropping`

`scPreferScaling`

`scPreferScalingAndCropping`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CompressMovies

ConvertToMovieJr

qtcompress

qtcompress.win

VideoProcessing

Declared In

`QuickTimeComponents.h`

TCFrameNumberToTimeCode

Converts a frame number into its corresponding timecode time value.

```
HandlerError TCFrameNumberToTimeCode (
   MediaHandler mh,
   long frameNumber,
   TimeCodeDef *tcdef,
   TimeCodeRecord *tcrec
);
```

Parameters

mh

The timecode media handler. You obtain this identifier by calling `GetMediaHandler` (page 1577).

frameNumber

The frame number that is to be converted.

tcdef

A pointer to the `TimeCodeDef` structure to use for the conversion.

tcrec

A pointer to the `TimeCodeRecord` structure that is to receive the time value.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

TimeCode Media Handlers

Declared In

`QuickTimeComponents.h`

TCGetCurrentTimeCode

Retrieves the timecode and source identification information for the current movie time.

```
HandlerError TCGetCurrentTimeCode (
   MediaHandler mh,
   long *frameNum,
   TimeCodeDef *tcdef,
   TimeCodeRecord *tcrec,
   UserData *srcRefH
);
```

Parameters

mh

The timecode media handler. You obtain this identifier by calling `GetMediaHandler` (page 1577).

frameNum

A pointer to a field that is to receive the current frame number. Set this field to `NIL` if you don't want to retrieve the frame number.

tcdef

A pointer to a `TimeCodeDef` structure. The media handler returns the movie's timecode definition information. Set this parameter to `NIL` if you don't want this information.

tcrec

A pointer to a `TimeCodeRecord` structure. The media handler returns the current time value. Set this parameter to `NIL` if you don't want this information.

srcRefH

A pointer to a field that is to receive a handle containing the source information as a `UserDataRecord` structure. It is your responsibility to dispose of this structure when you are done with it. Set this field to `NIL` if you don't want this information.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTKitTimeCode

qttimecode

qttimecode.win

Declared In
`QuickTimeComponents.h`

TCGetDisplayOptions

Retrieves the text characteristics that apply to timecode information displayed in a movie.

```
HandlerError TCGetDisplayOptions (
   MediaHandler mh,
   TCTextOptionsPtr textOptions
);
```

Parameters

mh

The timecode media handler. You obtain this identifier by calling `GetMediaHandler` (page 1577).

textOptions

A pointer to a `TCTextOptions` structure. This structure will receive font and `style` information.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTKitTimeCode

qttimecode

qttimecode.win

TimeCode Media Handlers

Declared In
`QuickTimeComponents.h`

TCGetSourceRef

Retrieves the source information from the timecode media sample reference.

```
HandlerError TCGetSourceRef (
   MediaHandler mh,
   TimeCodeDescriptionHandle tcdH,
   UserData *srefH
);
```

Parameters

mh

The timecode media handler. You obtain this identifier by calling `GetMediaHandler` (page 1577).

tcdH

Specifies a handle to a `TimeCodeDescription` structure that defines the media sample reference for this operation.

srefH

Specifies a pointer to a handle that will receive the source information as a `UserDataRecord` structure. It is your application's responsibility to dispose of this structure when you are done with it.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

TCGetTimeCodeAtTime

Returns a track's timecode information corresponding to a specific media time.

```
HandlerError TCGetTimeCodeAtTime (
   MediaHandler mh,
   TimeValue mediaTime,
   long *frameNum,
   TimeCodeDef *tcdef,
   TimeCodeRecord *tcdata,
   UserData *srcRefH
);
```

Parameters

mh

The timecode media handler. You obtain this identifier by calling `GetMediaHandler` (page 1577).

mediaTime

A time value for which you want to retrieve timecode information. This time value is expressed in the media's time coordinate system.

frameNum

A pointer to a field that is to receive the current frame number. Set this field to `NIL` if you don't want to retrieve the frame number.

tcdef

A pointer to a `TimeCodeDef` structure. The media handler returns the movie's timecode definition information. Set this parameter to `NIL` if you don't want this information.

tcdata

A pointer to a `TimeCodeRecord` structure. The media handler returns the current time value. Set this parameter to `NIL` if you don't want this information.

srcRefH

A pointer to a field that is to receive a handle containing the source information as a `UserDataRecord` structure. It is your responsibility to dispose of this structure when you are done with it. Set this field to `NIL` if you don't want this information.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

TimeCode Media Handlers

Declared In

`QuickTimeComponents.h`

TCGetTimeCodeFlags

Retrieves the timecode control flags.

```
HandlerError TCGetTimeCodeFlags (
   MediaHandler mh,
   long *flags
);
```

Parameters

mh

The timecode media handler. You obtain this identifier by calling `GetMediaHandler` (page 1577).

flags

A pointer to a field that is to receive a control flag (see below). See these constants:

`tcdfShowTimeCode`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

QTKitTimeCode

qttimecode

qttimecode.win

Declared In

`QuickTimeComponents.h`

TCSetDisplayOptions

Sets the text characteristics that apply to timecode information displayed in a movie.

```
HandlerError TCSetDisplayOptions (
   MediaHandler mh,
   TCTextOptionsPtr textOptions
);
```

Parameters

mh

The timecode media handler. You obtain this identifier by calling `GetMediaHandler` (page 1577).

textOptions

A pointer to a `TCTextOptions` structure. This structure contains font and `style` information.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code
QTKitTimeCode

qttimecode

qttimecode.win

Declared In
`QuickTimeComponents.h`

TCSetSourceRef

Changes the source information in the timecode media sample reference.

```
HandlerError TCSetSourceRef (
   MediaHandler mh,
   TimeCodeDescriptionHandle tcdH,
   UserData srefH
);
```

Parameters

mh

The timecode media handler. You obtain this identifier by calling `GetMediaHandler` (page 1577).

tcdH

Specifies a handle containing the timecode media sample reference that is to be updated.

srefH

Specifies a handle to the source information to be placed in the sample reference as a `UserDataRecord` structure. It is your application's responsibility to dispose of this structure when you are done with it.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTKitTimeCode

qttimecode

qttimecode.win

TimeCode Media Handlers

Declared In
`QuickTimeComponents.h`

TCSetTimeCodeFlags

Changes the flag that affects how the toolbox handles timecode information.

```
HandlerError TCSetTimeCodeFlags (
   MediaHandler mh,
   long flags,
   long flagsMask
);
```

Parameters

mh

The timecode media handler. You obtain this identifier by calling `GetMediaHandler` (page 1577).

flags

The new flag value. See these constants:
`tcdfShowTimeCode`

flagsMask

Specifies which of the flag values are to change. The media handler modifies only those flag values that correspond to bits that are set to 1 in this parameter. Use the flag values from the `flags` parameter. To turn off timecode display, set the `tcdfShowTimeCode` flag to 1 in the `flagsMask` parameter and to 0 in the `flags` parameter.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTKitTimeCode
qttimecode
qttimecode.win
TimeCode Media Handlers

Declared In
`QuickTimeComponents.h`

TCTimeCodeToFrameNumber

Converts a timecode time value into its corresponding frame number.

```
HandlerError TCTimeCodeToFrameNumber (
   MediaHandler mh,
   TimeCodeDef *tcdef,
   TimeCodeRecord *tcrec,
   long *frameNumber
);
```

Parameters

mh

The timecode media handler. You obtain this identifier by calling `GetMediaHandler` (page 1577).

tcdef

A pointer to the `TimeCodeDef` structure to use for the conversion.

tcrec
A pointer to the `TimeCodeRecord` structure containing the time value to convert.

frameNumber
A pointer to a field that is to receive the frame number that corresponds to the time value in the `tcrec` parameter.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTKitTimeCode

qttimecode

qttimecode.win

TimeCode Media Handlers

Declared In
`QuickTimeComponents.h`

TCTimeCodeToString

Converts a time value into a text string (HH:MM:SS:FF).

```
HandlerError TCTimeCodeToString (
   MediaHandler mh,
   TimeCodeDef *tcdef,
   TimeCodeRecord *tcrec,
   StringPtr tcStr
);
```

Parameters

mh
The timecode media handler. You obtain this identifier by calling `GetMediaHandler` (page 1577).

tcdef
A pointer to the `TimeCodeDef` structure to use for the conversion.

tcrec
A pointer to the `TimeCodeRecord` structure to use for the conversion.

tcStr
A pointer to a text string that is to receive the converted time value.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
If the timecode uses the dropframe technique, the separators are semicolons (;) rather than colons (:).

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTKitTimeCode
qttimecode
qttimecode.win
TimeCode Media Handlers

Declared In
`QuickTimeComponents.h`

TextExportGetDisplayData

Retrieves text display information for the current sample in the specified text export component.

```
ComponentResult TextExportGetDisplayData (
   TextExportComponent ci,
   TextDisplayData *textDisplay
);
```

Parameters

ci
: Specifies the text export component for this operation. Applications can obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

textDisplay
: Contains a pointer to a `TextDisplayData` structure. On return, this structure contains the display settings of the current text sample.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
You call this function to retrieve the text display data structure for a text sample. The text display data structure contains the formatting information for the text sample. When the text export component exports a text sample, it uses the information in this structure to generate the appropriate text descriptors for the sample. Likewise, when the text import component imports a text sample, it sets the appropriate fields in the text display data structure based on the sample's text descriptors.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

TextExportGetSettings

Retrieves the value of the text export option for the specified text export component.

```
ComponentResult TextExportGetSettings (
   TextExportComponent ci,
   long *setting
);
```

Parameters

ci
: Specifies the text export component for this operation. Applications can obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

setting
: Contains a pointer to a 32-bit integer. On return, this integer contains a constant (see below) that represents the current value of the text export option. See these constants:
  - `kMovieExportTextOnly`
  - `kMovieExportAbsoluteTime`
  - `kMovieExportRelativeTime`

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

TextExportGetTimeFraction

Retrieves the time scale the specified text export component uses to calculate time stamps.

```
ComponentResult TextExportGetTimeFraction (
   TextExportComponent ci,
   long *movieTimeFraction
);
```

Parameters

ci
: Specifies the text export component for this operation. Applications can obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

movieTimeFraction
: Contains a pointer to a 32-bit integer. On return, this integer contains the time scale used in the fractional part of time stamps.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
You call this function to retrieve the time scale used by the text export component to calculate the fractional part of time stamps. You set a text component's time scale by specifying it in the Text Export Settings dialog box or by calling `TextExportSetTimeFraction` (page 548).

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

TextExportSetSettings

Sets the value of the text export option for the specified text export component.

```
ComponentResult TextExportSetSettings (
   TextExportComponent ci,
   long setting
);
```

Parameters

`ci`
Specifies the text export component for this operation. Applications can obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

`setting`
A constant (see below) that specifies the new value of the text export option. See these constants:
`kMovieExportTextOnly`
`kMovieExportAbsoluteTime`
`kMovieExportRelativeTime`

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

TextExportSetTimeFraction

Sets the time scale the specified text export component uses to calculate time stamps.

```
ComponentResult TextExportSetTimeFraction (
   TextExportComponent ci,
   long movieTimeFraction
);
```

Parameters

ci

Specifies the text export component for this operation. Applications can obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

movieTimeFraction

Specifies the time scale used in the fractional part of time stamps. The value should be between 1 and 10000, inclusive.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

You call this function to set the time scale used by the text export component to calculate the fractional part of time stamps. You can also set a text component's time scale by specifying it in the text export settings dialog box. You can retrieve a text component's time scale by calling `TextExportGetTimeFraction` (page 547).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

TweenerDoTween

Performs a tween operation.

```
ComponentResult TweenerDoTween (
   TweenerComponent tc,
   TweenRecord *tr
);
```

Parameters

tc

The tween component for this operation.

tr

A pointer to the `TweenRecord` structure for the tween operation.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

QuickTime calls this function to interpolate the data used during a tween operation. The `TweenRecord` structure contains complete information about the tween operation, including the start and end values for the operation and a percentage that indicates the progress towards completion of the tween sample. This function should use the information in the tween record to calculate the tweened value, and should call the data function specified in the tween record, passing it the tweened value.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

TweenerInitialize

Initializes your tween component for a single tween operation.

```
ComponentResult TweenerInitialize (
   TweenerComponent tc,
   QTAtomContainer container,
   QTAtom tweenAtom,
   QTAtom dataAtom
);
```

Parameters

tc
The tween component for this operation.

container
The container that holds the atoms specified by the `tweenAtom` and `dataAtom` parameters.

tweenAtom
The atom that contains all parameters for defining this tween. This includes the data atom and any special atoms, such as an atom of type `'qdrg'`, that may be necessary.

dataAtom
The atom that contains the values to be tweened. This atom is a child of the atom specified by the `tweenAtom` parameter. This parameter is provided as a convenience; you can also call QT atom container functions to locate the data atom in the container.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function sets up the tween component when it is first used. In your implementation of this function, you can allocate storage and set up any structures that you need for the duration of a tween operation. Although the container that holds the data atom is available during each call to `TweenerDoTween` (page 549), you can improve the performance of your tween component by extracting the data to be used by the `TweenerDoTween` function in this function.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

TweenerReset

Cleans up when the tween operation is finished.

```
ComponentResult TweenerReset (
   TweenerComponent tc
);
```

Parameters

tc

The tween component for this operation.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function releases storage allocated by the tween component when the component is no longer being used. It should release any storage allocated by the `TweenerInitialize` (page 550) function and close or release any other resources used by the component. A tween component may receive a `TweenerInitialize` call after being reset.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

Callbacks

MovieExportGetDataProc

Defines a data source for an export operation.

```
typedef OSErr (*MovieExportGetDataProcPtr) (void *refCon, MovieExportGetDataParams
 *params);
```

If you name your function `MyMovieExportGetDataProc`, you would declare it this way:

```
OSErr MyMovieExportGetDataProc (
    void                        *refCon,
    MovieExportGetDataParams    *params );
```

Parameters

refCon

Contains the value passed to `MovieExportAddDataSource` (page 466) in the `refCon` parameter

params

The sample request is made through a `MovieExportGetDataParams` structure.

Return Value

See `Error Codes`. Your callback should return `noErr` if there is no error.

Discussion

This callback is passed to `MovieExportAddDataSource` (page 466) to define a new data source for an export operation. The function is used by the exporting application to request source media data to be used in the export operation. For example, in a video export operation, frames of video data (either compressed or uncompressed) are provided. In a sound export operation, buffers of audio (either compressed or uncompressed) are provided.

Special Considerations

The data pointed to by `dataPtr` must remain valid until the next call to this function. The `MovieExportGetDataProc` callback is responsible for allocating and disposing of the memory associated with this data pointer.

Declared In

`QuickTimeComponents.h`

MovieExportGetPropertyProc

Returns parameters that determine the appropriate format for movie export data.

```
typedef OSErr (*MovieExportGetPropertyProcPtr) (void *refcon, long trackID, OSType
 propertyType, void *propertyValue);
```

If you name your function `MyMovieExportGetPropertyProc`, you would declare it this way:

```
OSErr MyMovieExportGetPropertyProc (
    void      *refcon,
    long      trackID,
    OSType    propertyType,
    void      *propertyValue );
```

Parameters

refcon

Contains the value passed to `MovieExportAddDataSource` (page 466) in the `refCon` parameter.

trackID

Specifies the value returned from `MovieExportAddDataSource` (page 466).

propertyType

Contains a pointer to the location of the requested property information.

propertyValue

Specifies the property being requested (see below). See these constants:

`scSoundSampleRateType`
`scSoundSampleSizeType`
`scSoundChannelCountType`
`scSoundCompressionType`
`movieExportWidth`
`movieExportHeight`
`movieExportVideoFilter`
`scSpatialSettingsType`
`scTemporalSettingsType`
`scDataRateSettingsType`
`movieExportDuration`

Return Value
See `Error Codes`. Your callback should return `noErr` if there is no error. If this function doesn't have a setting for a requested property, it should return an error.

Discussion
This function is passed to `MovieExportAddDataSource` (page 466) to define a new data source for an export operation. For example, a video export operation may call this function to determine the dimensions of the destination video track. The export component provides a default value for the property based on the source data format. For example, if no values for video track width and height properties were provided by the callback function, the dimensions of the source data would be used.

Declared In
`QuickTimeComponents.h`

SCModalFilterProc

Filter routine called when a user event occurs in a sequence compression modal dialog box.

```
typedef Boolean (*SCModalFilterProcPtr) (DialogPtr theDialog, EventRecord *theEvent,
 short *itemHit, long refcon);
```

If you name your function `MySCModalFilterProc`, you would declare it this way:

```
Boolean MySCModalFilterProc (
    DialogPtr      theDialog,
    EventRecord    *theEvent,
    short          *itemHit,
    long           refcon );
```

Parameters

theDialog

A pointer to a dialog box.

theEvent

A pointer to an `EventRecord` structure that defines a user event.

itemHit

A pointer to an item ID number in the dialog box.

refcon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

Return Value

Return TRUE if the event was handled, FALSE otherwise.

Declared In

`QuickTimeComponents.h`

SCModalHookProc

Called whenever the user selects an item in the dialog box.

```
typedef short (*SCModalHookProcPtr) (DialogPtr theDialog, short itemHit, void
*params, long refcon);
```

If you name your function `MySCModalHookProc`, you would declare it this way:

```
short MySCModalHookProc (
    DialogPtr    theDialog,
    short        itemHit,
    void         *params,
    long         refcon );
```

Parameters

theDialog

A pointer to a dialog box.

itemHit

A pointer to an item ID number in the dialog box.

params

A pointer to your data area.

refcon

A reference constant that the client code supplies to your callback.

Return Value

Return TRUE if the event was handled, FALSE otherwise.

Discussion

You can use this callback to customize the operation of the standard image-compression dialog box. For example, you might want to support a custom button that activates a secondary dialog box. Another possibility would be to provide additional validation support when the user clicks OK.

Declared In

`QuickTimeComponents.h`

Data Types

GraphicImageMovieImportComponent

Represents a type used by the Movie Components API.

```
typedef ComponentInstance GraphicImageMovieImportComponent;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

HandlerError

Represents a type used by the Movie Components API.

```
typedef ComponentResult HandlerError;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

MovieExportComponent

Represents a type used by the Movie Components API.

```
typedef ComponentInstance MovieExportComponent;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

MovieExportGetDataUPP

Represents a type used by the Movie Components API.

```
typedef STACK_UPP_TYPE(MovieExportGetDataProcPtr) MovieExportGetDataUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

MovieExportGetPropertyUPP

Represents a type used by the Movie Components API.

```
typedef STACK_UPP_TYPE(MovieExportGetPropertyProcPtr) MovieExportGetPropertyUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

MovieImportComponent

Represents a type used by the Movie Components API.

```
typedef ComponentInstance MovieImportComponent;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

pnotComponent

Represents a type used by the Movie Components API.

```
typedef ComponentInstance pnotComponent;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SCModalFilterUPP

Represents a type used by the Movie Components API.

```
typedef STACK_UPP_TYPE(SCModalFilterProcPtr) SCModalFilterUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SCModalHookUPP

Represents a type used by the Movie Components API.

```
typedef STACK_UPP_TYPE(SCModalHookProcPtr) SCModalHookUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SCParams

Provides data for the SCGetCompressionExtended function.

```
struct SCParams {
     long                flags;
     CodecType           theCodecType;
     CodecComponent      theCodec;
     CodecQ              spatialQuality;
     CodecQ              temporalQuality;
     short               depth;
     Fixed               frameRate;
     long                keyFrameRate;
     long                reserved1;
     long                reserved2;
 };
```

Fields
`flags`

Discussion
Flags (see below). See these constants:

- `scGetCompression`
- `scShowMotionSettings`
- `scSettingsChangedItem`

`theCodecType`

Discussion
A compressor type; see `Codec Identifiers`.

`theCodec`

Discussion
An instance of a compressor component, obtained by calling `OpenComponent` or `OpenDefaultComponent`.

`spatialQuality`

Discussion
Constants (see below) that determine image spatial quality. See these constants:

- `codecMinQuality`
- `codecLowQuality`
- `codecNormalQuality`
- `codecHighQuality`
- `codecMaxQuality`
- `codecLosslessQuality`

`temporalQuality`

Discussion
Constants (see below) that determine image temporal quality.

`depth`

Discussion
Image data depth.

`frameRate`

Discussion
The frame rate.

`keyFrameRate`

Discussion
The key frame rate.

`reserved1`

Discussion
Reserved.

`reserved2`

Discussion
Reserved.

Related Functions
`SCGetCompressionExtended` (page 525)

Declared In
`QuickTimeComponents.h`

TCTextOptions

Holds text font and style information.

```
struct TCTextOptions {
     short       txFont;
     short       txFace;
     short       txSize;
     short       pad;
     RGBColor    foreColor;
     RGBColor    backColor;
 };
```

Fields
`txFont`

Discussion
Specifies the number of the font.

`txFace`

Discussion
Specifies the font's `style` (bold, italic, and so on).

`txSize`

Discussion
Specifies the font's size.

`pad`

Discussion
Unused field to make structure long-word aligned.

`foreColor`

Discussion
Specifies the foreground color.

`backColor`

Discussion
Specifies the background color.

Related Functions
`TCGetDisplayOptions` (page 539)
`TCSetDisplayOptions` (page 542)

Declared In
`QuickTimeComponents.h`

TCTextOptionsPtr

Represents a type used by the Movie Components API.

```
typedef TCTextOptions * TCTextOptionsPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

TextDisplayData

Contains formatting information for a text sample.

```
struct TextDisplayData {
     long         displayFlags;
     long         textJustification;
     RGBColor     bgColor;
     Rect         textBox;
     short        beginHilite;
     short        endHilite;
     RGBColor     hiliteColor;
     Boolean      doHiliteColor;
     SInt8        filler;
     TimeValue    scrollDelayDur;
     Point        dropShadowOffset;
     short        dropShadowTransparency;
 };
```

Fields

`displayFlags`

Discussion
Contains flags (see below) that represent the values of text descriptors. See these constants:

`dfDontDisplay`

`dfDontAutoScale`

`dfClipToTextBox`

`dfShrinkTextBoxToFit`

`dfScrollIn`

`dfScrollOut`

`dfHorizScroll`

`dfReverseScroll`

`textJustification`

Discussion
Contains constants (see below) that specify the alignment of the text in the text box. Possible values are `teFlushDefault`, `teCenter`, `teFlushRight`, and `teFlushLeft`. For more information on text alignment and the text justification constants, see the "TextEdit" chapter of *Inside Macintosh: Text*. See these constants:

`bgColor`

Discussion
Specifies the background color of the rectangle specified by the `textBox` field. The background color is specified as an RGB color value.

`textBox`

Discussion
Specifies the rectangle of the text box.

`beginHilite`

Discussion
Specifies the one-based index of the first character in the sample to highlight.

`endHilite`

Discussion
Specifies the one-based index of the last character in the sample to highlight.

`doHiliteColor`

Discussion
Specifies whether to use the color specified by the `hiliteColor` field for highlighting. If the `value` of this field is TRUE, the highlight color is used for highlighting. If the `value` of this field is FALSE, reverse video is used for highlighting.

`filler`

Discussion
Reserved.

`scrollDelayDur`

Discussion
Specifies a scroll delay. The scroll delay is specified as the number of units of delay in the text track's time scale. For example, if the time scale is 600, a scroll delay of 600 causes the sample text to be delayed one second. In order for this field to take effect, scrolling must be enabled.

`dropShadowOffset`

Discussion
Specifies an offset for the drop shadow. For example, if the point specified is (3,4), the drop shadow is offset 3 pixels to the right and 4 pixels down. In order for this field to take effect, drop shadowing must be enabled.

`dropShadowTransparency`

Discussion
Specifies the intensity of the drop shadow as a value between 0 and 255. In order for this field to take effect, drop shadowing must be enabled.

Discussion
When the text export component exports a text sample, it uses the information in this structure to generate the appropriate text descriptors for the sample. Likewise, when the text import component imports a text sample, it sets the appropriate fields in this structure based on the sample's text descriptors.

Related Functions
`TextExportGetDisplayData` (page 546)

Declared In
`QuickTimeComponents.h`

TextExportComponent

Represents a type used by the Movie Components API.

```
typedef ComponentInstance TextExportComponent;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

TimeCodeDef

Contains timecode format information.

```
struct TimeCodeDef {
     long          flags;
     TimeScale     fTimeScale;
     TimeValue     frameDuration;
     UInt8         numFrames;
     UInt8         padding;
 };
```

Fields

`flags`

Discussion

Contains flags (see below) that provide timecode format information. See these constants:

`tcDropFrame`

`tc24HourMax`

`tcNegTimesOK`

`tcCounter`

`fTimeScale`

Discussion

Contains the time scale for interpreting the `frameDuration` field. This field indicates the number of time units per second.

`frameDuration`

Discussion

Specifies how long each frame lasts, in the units defined by the `fTimeScale` field.

`numFrames`

Discussion

Indicates the number of frames stored per second. In the case of timecodes that are interpreted as counters, this field indicates the number of frames stored per timer "tick."

`padding`

Discussion

Unused.

Related Functions

`TCFrameNumberToTimeCode` (page 538)
`TCGetCurrentTimeCode` (page 538)
`TCGetTimeCodeAtTime` (page 540)
`TCTimeCodeToFrameNumber` (page 544)
`TCTimeCodeToString` (page 545)

Declared In

`QuickTimeComponents.h`

TimeCodeDescriptionHandle

Represents a type used by the Movie Components API.

```
typedef TimeCodeDescriptionPtr * TimeCodeDescriptionHandle;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

TimeCodeDescriptionPtr

Represents a type used by the Movie Components API.

```
typedef TimeCodeDescription * TimeCodeDescriptionPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

TimeCodeRecord

Interprets time information as both a time value (HH:MM:SS:FF) and a frame count.

```
union TimeCodeRecord {
     TimeCodeTime       t;
     TimeCodeCounter    c;
 };
```

Fields
`t`

Discussion
The timecode value interpreted as time in a `TimeCodeTime` structure.

`c`

Discussion
The timecode value interpreted as a frame count in a `TimeCodeCounter` structure.

Discussion
When you use the timecode media handler to work with time values, the media handler uses `TimeCodeRecord` structures to store the time values. These structures allows you to interpret the time information as either a time value (HH:MM:SS:FF) or a counter value. Given a timecode definition, you can freely convert from frame numbers to time values and from time values to frame numbers. For a time value of 00:00:12:15 (HH:MM:SS:FF), you would obtain a frame number of 375 ((12*30) +15).

Related Functions
`TCFrameNumberToTimeCode` (page 538)
`TCGetCurrentTimeCode` (page 538)
`TCGetTimeCodeAtTime` (page 540)
`TCTimeCodeToFrameNumber` (page 544)
`TCTimeCodeToString` (page 545)

Declared In
`QuickTimeComponents.h`

TweenerComponent

Represents a type used by the Movie Components API.

```
typedef ComponentInstance TweenerComponent;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

TweenRecord

Passes information to your tween component's TweenDoTween method.

```
struct TweenRecord {
     long                version;
     QTAtomContainer     container;
     QTAtom              tweenAtom;
     QTAtom              dataAtom;
     Fixed               percent;
     TweenerDataUPP      dataProc;
     void *              private1;
     void *              private2;
 };
```

Fields
`version`

Discussion
The version number of this structure. This field is initialized to 0.

`container`

Discussion
The atom container that contains the tween data.

`tweenAtom`

Discussion
The atom for this tween entry's data in the container.

`percent`

Discussion
The percentage by which to change the data.

`dataProc`

Discussion
The procedure the tween component calls to send the tweened value to the receiving track.

`private1`

Discussion
Reserved.

`private2`

Discussion
Reserved.

Related Functions

`TweenerDataProc`

`TweenerDoTween` (page 549)

Declared In

`QuickTimeComponents.h`

Constants

MIDIImportSetSettings Values

Constants passed to MIDIImportSetSettings.

```
enum {
  kMIDIImportSilenceBefore      = 1 << 0,
  kMIDIImportSilenceAfter       = 1 << 1,
  kMIDIImport20Playable         = 1 << 2,
  kMIDIImportWantLyrics         = 1 << 3
};
```

Declared In

`QuickTimeComponents.h`

TextExportSetSettings Values

Constants passed to TextExportSetSettings.

```
enum {
  kMovieExportTextOnly          = 0,
  kMovieExportAbsoluteTime      = 1,
  kMovieExportRelativeTime      = 2
};
```

Declared In

`QuickTimeComponents.h`

movieExportDuration

Constants grouped with movieExportDuration.

```
enum {
  movieExportUseConfiguredSettings = 'ucfg', /* pointer to Boolean*/
  movieExportWidth                = 'wdth', /* pointer to Fixed*/
  movieExportHeight               = 'hegt', /* pointer to Fixed*/
  movieExportDuration             = 'dura', /* pointer to TimeRecord*/
  movieExportVideoFilter          = 'iflt', /* pointer to QTAtomContainer*/
  movieExportTimeScale            = 'tmsc' /* pointer to TimeScale*/
};
```

Constants

`movieExportWidth`

A fixed integer that represents a video track's image width in pixels.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeComponents.h`.

`movieExportHeight`

A fixed integer that represents a video track's image height in pixels.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeComponents.h`.

`movieExportDuration`

The `TimeRecord` structure for the whole movie.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeComponents.h`.

`movieExportVideoFilter`

A pointer to a `QTAtomContainer` handle that references a video track's filter atom container.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeComponents.h`.

Declared In

`QuickTimeComponents.h`

MovieImportDataRef Values

Constants passed to MovieImportDataRef.

```
enum {
  movieImportCreateTrack        = 1,
  movieImportInParallel         = 2,
  movieImportMustUseTrack       = 4,
  movieImportWithIdle           = 16
};
enum {
  movieImportResultUsedMultipleTracks = 8,
  movieImportResultNeedIdles    = 32,
  movieImportResultComplete     = 64
};
```

Constants

`movieImportResultNeedIdles`

Undocumented

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeComponents.h`.

Declared In
`QuickTimeComponents.h`

Standard Compression Constants

Constants that represent constants for Standard Compression.

```
enum {
  /*
   * Indicates the client is ready to use the ICM compression session
   * API to perform compression operations. StdCompression disables
   * frame reordering and multi pass encoding if this flag is cleared.
   */
  scAllowEncodingWithCompressionSession = 1L << 8,
  /*
   * Indicates the client does not want the user to change the frame
   * reordering setting.
   */
  scDisableFrameReorderingItem  = 1L << 9,
  /*
   * Indicates the client does not want the user to change the multi
   * pass encoding setting
   */
  scDisableMultiPassEncodingItem = 1L << 10
};
enum {
  /*
   * Specifies if frame reordering can occur in encoding.
   */
  scVideoAllowFrameReorderingType = 'bfra', /* pointer to Boolean*/
  /*
   * The settings to control multi pass encoding.
   */
  scVideoMultiPassEncodingSettingsType = 'mpes' /* pointer to
SCVideoMutiPassEncodingSettings struct*/
};
enum {
  scListEveryCodec              = 1L << 1,
  scAllowZeroFrameRate          = 1L << 2,
  scAllowZeroKeyFrameRate       = 1L << 3,
  scShowBestDepth               = 1L << 4,
  scUseMovableModal             = 1L << 5,
  scDisableFrameRateItem        = 1L << 6,
  scShowDataRateAsKilobits      = 1L << 7
};
enum {
  scOKItem                      = 1,
  scCancelItem                  = 2,
  scCustomItem                  = 3
};
enum {
  scPositionRect                = 2,
  scPositionDialog              = 3,
  scSetTestImagePictHandle      = 4,
  scSetTestImagePictFile        = 5,
  scSetTestImagePixMap          = 6,
  scGetBestDeviceRect           = 7,
  scRequestImageSettings        = 10,
  scCompressImage               = 11,
  scCompressPicture             = 12,
  scCompressPictureFile         = 13,
  scRequestSequenceSettings     = 14,
  scCompressSequenceBegin       = 15,
  scCompressSequenceFrame       = 16,
  scCompressSequenceEnd         = 17,
  scDefaultPictHandleSettings   = 18,
  scDefaultPictFileSettings     = 19,
  scDefaultPixMapSettings       = 20,
  scGetInfo                     = 21,
  scSetInfo                     = 22,
  scNewGWorld                   = 23
};
enum {
  scPreferCropping              = 1 << 0,
  scPreferScaling               = 1 << 1,
  scPreferScalingAndCropping    = scPreferScaling | scPreferCropping,
  scDontDetermineSettingsFromTestImage = 1 << 2
};
enum {
  scSpatialSettingsType         = 'sptl', /* pointer to SCSpatialSettings struct*/
  scTemporalSettingsType       = 'tprl', /* pointer to SCTemporalSettings struct*/
  scDataRateSettingsType       = 'drat', /* pointer to SCDataRateSettings struct*/
  scColorTableType              = 'clut', /* pointer to CTabHandle*/
  scProgressProcType            = 'prog', /* pointer to ProgressRecord struct*/
  scExtendedProcsType           = 'xprc', /* pointer to SCExtendedProcs struct*/
  scPreferenceFlagsType         = 'pref', /* pointer to long*/
  scSettingsStateType           = 'ssta', /* pointer to Handle*/
  scSequenceIDType              = 'sequ', /* pointer to ImageSequence*/
  scWindowPositionType          = 'wndw', /* pointer to Point*/
  scCodecFlagsType              = 'cflg', /* pointer to CodecFlags*/
  scCodecSettingsType           = 'cdec', /* pointer to Handle*/
  scForceKeyValueType           = 'ksim', /* pointer to long*/
  scCompressionListType         = 'ctyl', /* pointer to OSType Handle*/
  scCodecManufacturerType       = 'cmfr', /* pointer to OSType*/
  scAvailableCompressionListType = 'avai', /* pointer to OSType Handle*/
  scWindowOptionsType           = 'shee', /* pointer to SCWindowSettings struct*/
  scSoundVBRCompressionOK       = 'cvbr', /* pointer to Boolean*/
  scSoundSampleRateChangeOK     = 'rcok', /* pointer to Boolean*/
  scSoundCompressionType        = 'ssct', /* pointer to OSType*/
  scSoundSampleRateType         = 'ssrt', /* pointer to UnsignedFixed*/
  scSoundInputSampleRateType    = 'ssir', /* pointer to UnsignedFixed*/
  scSoundSampleSizeType         = 'ssss', /* pointer to short*/
  scSoundChannelCountType       = 'sscc' /* pointer to short*/
};
enum {
  scTestImageWidth              = 80,
  scTestImageHeight             = 80
};
enum {
  scUserCancelled               = 1
};
enum {
  scWindowRefKindCarbon         = 'carb' /* WindowRef*/
};
```

Constants

`scVideoAllowFrameReorderingType`

Pointer to Boolean.

Available in Mac OS X v10.3 and later.

Declared in `QuickTimeComponents.h`.

`scSpatialSettingsType`
: A video track's `SCSpatialSettings` structure.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`scTemporalSettingsType`
: A video track's `SCTemporalSettings` structure.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`scDataRateSettingsType`
: A video track's `SCDataRateSettings` structure.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`scCodecSettingsType`
: Pointer to `Handle`.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`scForceKeyValueType`
: Pointer to long.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`scCodecManufacturerType`
: Pointer to `OSType`.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`scAvailableCompressionListType`
: Pointer to `OSType` `Handle`.

  Available in Mac OS X v10.2 and later.

  Declared in `QuickTimeComponents.h`.

`scWindowOptionsType`
: Pointer to `SCWindowSettings` struct.

  Available in Mac OS X v10.3 and later.

  Declared in `QuickTimeComponents.h`.

`scSoundVBRCompressionOK`
: Pointer to Boolean.

  Available in Mac OS X v10.2 and later.

  Declared in `QuickTimeComponents.h`.

`scSoundSampleRateChangeOK`
: Pointer to Boolean.

  Available in Mac OS X v10.2 and later.

  Declared in `QuickTimeComponents.h`.

`scSoundCompressionType`
: A sound track's compression type constant; see `Codec Identifiers`.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`scSoundSampleRateType`
: An `UnsignedFixed` value that represents a sound track's sampling rate.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`scSoundInputSampleRateType`
: Pointer to `UnsignedFixed`.

  Available in Mac OS X v10.2 and later.

  Declared in `QuickTimeComponents.h`.

`scSoundSampleSizeType`
: A short integer that represents a sound track's sample size.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`scSoundChannelCountType`
: A short integer that represents a sound track's channel count.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

Declared In
`QuickTimeComponents.h`

SCSetCompressFlags Values

Constants passed to SCSetCompressFlags.

```
enum {
  scCompressFlagIgnoreIdenticalFrames = 1
};
```

Declared In
`QuickTimeComponents.h`

SCParams Values

Constants passed to SCParams.

```
enum {
  scGetCompression              = 1,
  scShowMotionSettings          = 1L << 0,
  scSettingsChangedItem         = -1
};
```

Constants

`scGetCompression`
: *Undocumented*

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`scShowMotionSettings`
: *Undocumented*

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

Declared In

`QuickTimeComponents.h`

TCSetTimeCodeFlags Values

Constants passed to TCSetTimeCodeFlags.

```
enum {
  tcdfShowTimeCode              = 1 << 0
};
```

Declared In

`QuickTimeComponents.h`

TimeCodeDef Values

Constants passed to TimeCodeDef.

```
enum {
  tcDropFrame                   = 1 << 0,
  tc24HourMax                   = 1 << 1,
  tcNegTimesOK                  = 1 << 2,
  tcCounter                     = 1 << 3
};
```

Constants

`tcDropFrame`
: Indicates that the timecode drops frames occasionally to stay in synchronization. Some timecodes run at other than a whole number of frames per second. For example, NTSC video runs at 29.97 frames per second. In order to resynchronize between the timecode rate and a 30 frames-per-second playback rate, the timecode drops a frame at a predictable time (in much the same way that leap years keep the calendar synchronized).

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`tc24HourMax`

Indicates that the timecode values return to 0 at 24 hours.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeComponents.h`.

`tcNegTimesOK`

Indicates that the timecode supports negative time values.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeComponents.h`.

Declared In

`QuickTimeComponents.h`

Compression and Decompression Reference for QuickTime

Framework:	Frameworks/QuickTime.framework
Declared in	ImageCompression.h

Overview

QuickTime compression and decompression APIs help applications compress and decompress movie data.

Functions by Task

Aligning Windows

`AlignScreenRect` (page 587)
: Aligns a specified rectangle to the strictest screen that the rectangle intersects.

`AlignWindow` (page 588)
: Moves a specified window to the nearest optimal alignment position.

`DragAlignedGrayRgn` (page 629)
: Drags a specified gray region along an optimal alignment grid.

`DragAlignedWindow` (page 630)
: Drags the specified window along an optimal alignment grid.

Applying Matrix Transformations

`TransformFixedPoints` (page 734)
: Transforms a set of fixed points through a specified matrix.

`TransformFixedRect` (page 734)
: Transforms the upper-left and lower-right points of a rectangle through a matrix that is specified by fixed points.

`TransformPoints` (page 735)
: Transforms a set of QuickDraw points through a specified matrix.

`TransformRect` (page 736)
: Transforms the upper-left and lower-right points of a rectangle through a specified matrix.

`TransformRgn` (page 737)
: Applies a specified matrix to a region.

Changing Sequence-Compression Parameters

`GetCSequenceMaxCompressionSize` (page 658)
Determines the maximum size an image will be after compression for a given compression sequence.

`GetCSequencePrevBuffer` (page 659)
Determines the location of the previous image buffer allocated by the compressor.

`SetCSequenceFlushProc` (page 716)
Assigns a data-unloading function to a sequence.

`SetCSequenceKeyFrameRate` (page 718)
Adjusts the key frame rate for the current sequence.

`SetCSequencePreferredPacketSize` (page 718)
Sets the preferred packet size for a sequence.

`SetCSequencePrev` (page 719)
Allows the application to set the pixel map and boundary rectangle used by the previous frame in temporal compression.

`SetCSequenceQuality` (page 720)
Adjusts the spatial or temporal quality for the current sequence.

Changing Sequence-Decompression Parameters

`GetDSequenceImageBuffer` (page 660)
Determines the location of the offscreen image buffer allocated by a decompressor.

`GetDSequenceScreenBuffer` (page 662)
Determines the location of the offscreen screen buffer allocated by a decompressor.

`PtInDSequenceData` (page 695)
Tests to see if a compressed image contains data at a a given point.

`SetDSequenceAccuracy` (page 721)
Adjusts the decompression accuracy for the current sequence.

`SetDSequenceDataProc` (page 722)
Assigns a data-loading function to a sequence.

`SetDSequenceMask` (page 724)
Assigns a clipping region to a sequence.

`SetDSequenceMatrix` (page 724)
Assigns a mapping matrix to a sequence.

`SetDSequenceMatte` (page 725)
Assigns a blend matte to a sequence.

`SetDSequenceSrcRect` (page 727)
Defines the portion of an image to decompress.

`SetDSequenceTimeCode` (page 728)
Sets the timecode value for a frame that is about to be decompressed.

`SetDSequenceTransferMode` (page 728)
Sets the mode used when drawing a decompressed image.

Constraining Compressed Data

`GetCSequenceDataRateParams` (page 657)
  Obtains the data rate parameters previously set with SetCSequenceDataRateParams.

`SetCSequenceDataRateParams` (page 715)
  Communicates information to compressors that can constrain compressed data in a particular sequence to a specific data rate.

Controlling Hardware Scaling

`GDGetScale` (page 649)
  Returns the current scale of the given screen graphics device.

`GDHasScale` (page 649)
  Returns the closest possible scaling that a particular screen device can be set to in a given pixel depth.

`GDSetScale` (page 650)
  Sets a screen graphics device to a new scale.

Creating an Effect Sample Description

`MakeImageDescriptionForEffect` (page 689)
  Returns an ImageDescription structure you can use to help create a sample description for an effect.

Creating File Previews

`AddFilePreview` (page 586)
  Adds a preview to a file.

`MakeFilePreview` (page 688)
  Creates a preview for a file.

Getting Information About Compressed Data

`GetCompressedImageSize` (page 653)
  Determines the size, in bytes, of a compressed image.

`GetCompressionTime` (page 655)
  Determines the estimated amount of time required to compress a given image.

`GetMaxCompressionSize` (page 670)
  Determines the maximum size an image will be after compression.

`GetSimilarity` (page 673)
  Compares a compressed image to a picture stored in a pixel map and returns a value indicating the relative similarity of the two images.

Getting Information About Compressor Components

`CodecManagerVersion` (page 600)
Determines the version of the installed Image Compression Manager.

`DisposeCodecNameList` (page 628)
Disposes of the compressor name list structure you obtained by calling GetCodecNameList.

`FindCodec` (page 645)
Determines which of the installed compressors or decompressors has been chosen to field requests made by using one of the special compressor identifiers.

`GetCodecInfo` (page 652)
Returns information about a single compressor component.

`GetCodecNameList` (page 652)
Retrieves a list of installed compressor components or types.

Image Compression Manager Utility Functions

`ICMDecompressComplete` (page 675)
Signals the completion of a decompression operation.

`ICMShieldSequenceCursor` (page 681)
Hides the cursor during decompression operations.

Image Transcoder Support

`ImageTranscodeDisposeFrameData` (page 685)
Disposes transcoded image data.

`ImageTranscodeFrame` (page 685)
Transcodes a frame of image data.

`ImageTranscodeSequenceBegin` (page 686)
Initiates an image transcoder sequence operation.

`ImageTranscodeSequenceEnd` (page 687)
Ends an image transcoder sequence operation.

Making Thumbnail Pictures

`MakeThumbnailFromPicture` (page 691)
Creates a thumbnail picture from a specified Picture structure.

`MakeThumbnailFromPictureFile` (page 692)
Creates a thumbnail picture from a specified picture file.

`MakeThumbnailFromPixMap` (page 693)
Creates a thumbnail picture from a specified PixMap structure.

Managing Matrices

`ConcatMatrix` (page 616)
: Concatenates two matrices, combining the transformations described by both matrices into a single matrix.

`CopyMatrix` (page 618)
: Copies the contents of one matrix into another matrix.

`EqualMatrix` (page 636)
: Compares two matrices and returns a result that indicates whether the matrices are equal.

`GetMatrixType` (page 669)
: Obtains information about a matrix.

`InverseMatrix` (page 687)
: Creates a new matrix that is the inverse of a specified matrix.

`MapMatrix` (page 694)
: Alters an existing matrix so that it defines a transformation from one rectangle to another.

`RectMatrix` (page 710)
: Creates a matrix that performs the translate and scale operation described by the relationship between two rectangles.

`RotateMatrix` (page 712)
: Modifies the contents of a matrix so that it defines a rotation operation.

`ScaleMatrix` (page 713)
: Modifies the contents of a matrix so that it defines a scaling operation.

`SetIdentityMatrix` (page 729)
: Sets the contents of a matrix so that it performs no transformation.

`SkewMatrix` (page 731)
: Modifies the contents of a matrix so that it defines a skew transformation.

`TranslateMatrix` (page 737)
: Adds a translation value to a specified matrix.

Obtaining a Graphics Importer Instance

`GetGraphicsImporterForDataRef` (page 663)
: Locates and opens a graphics importer component that can be used to draw the image from specified data reference.

`GetGraphicsImporterForDataRefWithFlags` (page 664)
: Locates and opens a graphics importer component for a data reference with flags that control the search process.

`GetGraphicsImporterForFile` (page 665)
: Locates and opens a graphics importer component that can be used to draw a specified file.

Working With Graphics Devices and Graphics Worlds

`GetBestDeviceRect` (page 651)
: Selects the deepest of all available graphics devices, while treating 16-bit and 32-bit screens as having equal depth.

`NewImageGWorld` (page 694)
: Creates an offscreen graphics world.

Working With Image Descriptions

`AddImageDescriptionExtension` (page 586)
: Adds an extension to an ImageDescription structure.

`CountImageDescriptionExtensionType` (page 618)
: Counts the number of extensions of a given type in an ImageDescriptionHandle.

`GetImageDescriptionExtension` (page 668)
: Returns a new handle with the data from a specified image description extension.

`GetNextImageDescriptionExtensionType` (page 672)
: Retrieves an image description structure extension type.

`QTGetPixelFormatDepthForImageDescription` (page 696)
: For a given pixel format, returns the depth value that should be used in image descriptions.

`RemoveImageDescriptionExtension` (page 711)
: Removes a specified extension from an ImageDescription structure.

Working With Pictures and PICT Files

`CompressPicture` (page 607)
: Compresses a single-frame image stored as a picture structure and places the result in another picture.

`CompressPictureFile` (page 609)
: Compresses a single-frame image stored as a picture file and places the result in another picture file.

`DrawPictureFile` (page 633)
: Draws an image from a specified picture file in the current graphics port.

`DrawTrimmedPicture` (page 634)
: Draws an image that is stored as a picture into the current graphics port and trims that image to fit a specified region.

`DrawTrimmedPictureFile` (page 635)
: Draws an image that is stored as a picture file into the current graphics port and trims that image to fit a specified region.

`FCompressPicture` (page 639)
: Compresses a single-frame image stored as a picture structure and places the result in another picture, with added control over the compression process.

`FCompressPictureFile` (page 641)
: Compresses a single-frame image stored as a picture file and places the result in another picture file, with added control over the compression process.

`GetPictureFileHeader` (page 673)
: Extracts the picture frame and file header from a specified picture file.

Working With Pixel Maps

`CompressImage` (page 605)
: Compresses a single-frame image that is currently stored as a pixel map structure.

`ConvertImage` (page 616)
: Converts the format of a compressed image.

`DecompressImage` (page 619)
: Decompresses a single-frame image into a pixel map structure.

`FCompressImage` (page 637)
: Compresses a single-frame image that is currently stored as a pixel map structure, with added control over the compression process.

`FDecompressImage` (page 643)
: Decompresses a single-frame image into a pixel map structure, with added control over the decompression process.

`GetImageDescriptionCTable` (page 667)
: Gets the custom color table for an image.

`SetImageDescriptionCTable` (page 730)
: Updates the custom ColorTable structure for an image.

`TrimImage` (page 738)
: Adjusts a compressed image to the boundaries defined by a specified rectangle.

Working With Sequences

`CDSequenceBusy` (page 589)
: Checks the status of an asynchronous compression or decompression operation.

`CDSequenceChangedSourceData` (page 589)
: Notifies the compressor that the image source data has changed.

`CDSequenceDisposeDataSource` (page 590)
: Disposes of a data source.

`CDSequenceDisposeMemory` (page 590)
: Disposes of memory allocated by the codec.

`CDSequenceEnd` (page 591)
: Indicates the end of processing for an image sequence.

`CDSequenceEquivalentImageDescription` (page 591)
: Reports whether two image descriptions are the same.

`CDSequenceFlush` (page 593)
: Stops a decompression sequence, aborting processing of any queued frames.

`CDSequenceInvalidate` (page 594)
: Notifies the Image Compression Manager that the destination port for the given image decompression sequence has been invalidated.

CDSequenceNewDataSource (page 595)
Creates a new data source.

CDSequenceNewMemory (page 597)
Requests codec-allocated memory.

CDSequenceSetSourceData (page 598)
Sets data in a new frame to a specific data source.

CompressSequenceBegin (page 609)
Signals the beginning of the process of compressing a sequence of frames.

CompressSequenceFrame (page 612)
Compresses one of a sequence of frames.

DecompressSequenceBegin (page 621)
Obsolete. See DecompressSequenceBeginS.

DecompressSequenceBeginS (page 622)
Sends a sample image to a decompressor.

DecompressSequenceFrame (page 624)
Obsolete. See DecompressSequenceFrameS.

DecompressSequenceFrameS (page 625)
Queues a frame for decompression and specifies the size of the compressed data; new applications should use DecompressSequenceFrameWhen.

DecompressSequenceFrameWhen (page 626)
Queues a frame for decompression and specifies the time at which decompression will begin.

SetSequenceProgressProc (page 731)
Installs a progress procedure for a sequence.

Working With the StdPix Function

GetCompressedPixMapInfo (page 654)
Retrieves information about a compressed image.

SetCompressedPixMapInfo (page 714)
Stores information about a compressed image for StdPix.

StdPix (page 732)
Extends the grafProcs field of the CGrafPort structure to support compressed data, mattes, matrices, and pixel maps, letting you intercept image data in compressed form before it is decompressed and displayed.

Working With Video Fields

ImageFieldSequenceBegin (page 682)
Initiates an image field sequence operation and specifies the input and output data format.

ImageFieldSequenceEnd (page 683)
Ends an image field sequence operation.

ImageFieldSequenceExtractCombine (page 683)
Performs field operations on video data.

Supporting Functions

`CDSequenceEquivalentImageDescriptionS` (page 592)
  Undocumented

`CDSequenceGetDataSource` (page 594)
  Gets a data source for a decompression sequence.

`CDSequenceSetSourceDataQueue` (page 599)
  Sets a data queue as the source for a decompression sequence.

`CDSequenceSetTimeBase` (page 599)
  Sets a time base for a decompression sequence.

`CompAdd` (page 601)
  Undocumented

`CompCompare` (page 601)
  Undocumented

`CompDiv` (page 602)
  Undocumented

`CompFixMul` (page 603)
  Undocumented

`CompMul` (page 603)
  Undocumented

`CompMulDiv` (page 604)
  Undocumented

`CompMulDivTrunc` (page 604)
  Undocumented

`CompNeg` (page 605)
  Undocumented

`CompShift` (page 614)
  Undocumented

`CompSquareRoot` (page 615)
  Undocumented

`CompSub` (page 615)
  Undocumented

`FixExp2` (page 646)
  Undocumented

`FixLog2` (page 647)
  Undocumented

`FixMulDiv` (page 647)
  Undocumented

`FixPow` (page 648)
  Undocumented

`FracSinCos` (page 648)
  Undocumented

`GetCSequenceFrameNumber` (page 657)
  Returns the current frame number of the specified sequence.

`GetCSequenceKeyFrameRate` (page 658)
: Determines the current key frame rate of a sequence.

`GetDSequenceMatrix` (page 661)
: Gets the matrix that was specified for a decompression sequence by a call to SetDSequenceMatrix, or that was set at DecompressSequenceBegin.

`GetDSequenceNonScheduledDisplayDirection` (page 661)
: Returns the display direction for a decompress sequence.

`GetDSequenceNonScheduledDisplayTime` (page 662)
: Gets the display time for a decompression sequence.

`GetGraphicsImporterForFileWithFlags` (page 666)
: Locates and opens a graphics importer component for a file with flags that control the search process.

`HitTestDSequenceData` (page 674)
: Undocumented

`ICMDecompressCompleteS` (page 676)
: Undocumented

`ICMGetPixelFormatInfo` (page 677)
: Retrieves pixel format information.

`ICMSequenceGetChainMember` (page 677)
: Undocumented

`ICMSequenceGetInfo` (page 678)
: Gets multiprocessing properties for compression and decompression sequences.

`ICMSequenceLockBits` (page 679)
: Undocumented

`ICMSequenceSetInfo` (page 679)
: Sets multiprocessing properties for compression and decompression sequences.

`ICMSequenceUnlockBits` (page 680)
: Undocumented

`ICMSetPixelFormatInfo` (page 681)
: Lets you define your own pixel format.

`MakeImageDescriptionForPixMap` (page 690)
: Fills out an ImageDescription structure corresponding to a PixMap structure.

`QTGetFileNameExtension` (page 696)
: Gets the extension to a file name.

`QTGetPixelSize` (page 697)
: Returns the bits per pixel for a given pixel format.

`QTGetPixMapHandleGammaLevel` (page 698)
: Retrieves the current PixMap extension's gamma level setting.

`QTGetPixMapHandleRequestedGammaLevel` (page 698)
: Retrieves the current PixMap extension's requested gamma level setting.

`QTGetPixMapHandleRowBytes` (page 699)
: Gets the rowBytes value for a pixel map accessed by a handle.

`QTGetPixMapPtrGammaLevel` (page 699)
: Retrieves the current PixMap extension's gamma level setting.

`QTGetPixMapPtrRequestedGammaLevel` (page 700)
: Retrieves the current PixMap extension's gamma level setting.

`QTGetPixMapPtrRowBytes` (page 700)
: Gets the rowBytes value for a pixel map accessed by a pointer.

`QTNewGWorld` (page 701)
: Creates an offscreen graphics world that may have a non-Macintosh pixel format.

`QTNewGWorldFromPtr` (page 703)
: Wraps a graphics world and pixel map structure around an existing block of memory containing an image.

`QTSetPixMapHandleGammaLevel` (page 704)
: Sets the gamma level of a pixel map.

`QTSetPixMapHandleRequestedGammaLevel` (page 705)
: Sets the requested gamma level of a pixel map.

`QTSetPixMapHandleRowBytes` (page 706)
: Sets the rowBytes value for a pixel map accessed by a handle.

`QTSetPixMapPtrGammaLevel` (page 706)
: Sets the gamma level of a pixel map.

`QTSetPixMapPtrRequestedGammaLevel` (page 707)
: Sets the requested gamma level of a pixel map.

`QTSetPixMapPtrRowBytes` (page 708)
: Sets the rowBytes value for a pixel map accessed by a pointer.

`QTUpdateGWorld` (page 708)
: Changes the pixel depth, boundary rectangle, or color table for an existing offscreen graphics world with a non-Macintosh pixel format.

`QuadToQuadMatrix` (page 710)
: Defines a matrix that maps between four input points and four output points.

`ReplaceDSequenceImageDescription` (page 712)
: Undocumented

`SetCSequenceFrameNumber` (page 717)
: Informs the compressor in use for the specified sequence that frames are being compressed out of order.

`SetDSequenceFlags` (page 723)
: Sets data loading flags.

`SetDSequenceNonScheduledDisplayDirection` (page 726)
: Sets the display direction for a decompress sequence.

`SetDSequenceNonScheduledDisplayTime` (page 726)
: Sets the display time for a decompression sequence.

`UnsignedFixMulDiv` (page 739)
: Performs multiplications and divisions on unsigned fixed-point numbers.

Functions

AddFilePreview

Adds a preview to a file.

```
OSErr AddFilePreview (
   short resRefNum,
   OSType previewType,
   Handle previewData
);
```

Parameters

resRefNum
: The resource file for this operation. You must have opened this resource file with write permission. If there is a preview in the specified file, the Movie Toolbox replaces that preview with a new one.

previewType
: The resource type to be assigned to the preview. This type should correspond to the type of data stored in the preview. For example, if you have created a QuickDraw picture that you want to use as a preview for a file, you should set the `previewType` parameter to `'PICT'`.

previewData
: A handle to the preview data. For example, if the preview data is a picture, you would provide a picture handle.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
You must have created the preview data yourself. If the specified file already has a preview defined, the `AddFilePreview` function replaces it with the new preview.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTMusicToo

Declared In
`ImageCompression.h`

AddImageDescriptionExtension

Adds an extension to an ImageDescription structure.

```
OSErr AddImageDescriptionExtension (
   ImageDescriptionHandle desc,
   Handle extension,
   long idType
);
```

Parameters

desc

A handle to the `ImageDescription` structure to add the extension to.

extension

The handle containing the extension data.

idType

A four-byte signature identifying the type of data being added to the `ImageDescription`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function allows the application to add custom data to an `ImageDescriptionHandle`. This data could be specific to the compressor component referenced by the `ImageDescription` structure.

Special Considerations

The Image Compression Manager makes a copy of the data referred to by the `extension` parameter. Thus, your application should dispose its copy of the data when it is no longer needed.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

AlignScreenRect

Aligns a specified rectangle to the strictest screen that the rectangle intersects.

```
void AlignScreenRect (
   Rect *rp,
   ICMAlignmentProcRecordPtr alignmentProc
);
```

Parameters

rp

A pointer to a rectangle defined in global screen coordinates.

alignmentProc

Points to your own alignment behavior function. Set this parameter to `NIL` to use the standard behavior.

Discussion

For a specification of your alignment function, see `ICMAlignmentProc`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

AlignWindow

Moves a specified window to the nearest optimal alignment position.

```
void AlignWindow (
   WindowRef wp,
   Boolean front,
   const Rect *alignmentRect,
   ICMAlignmentProcRecordPtr alignmentProc
);
```

Parameters

wp
Points to the window to be aligned.

front
The frontmost window. If the `front` parameter is TRUE and the window specified in the `wp` parameter isn't the active window, `AlignWindow` makes it the active window.

alignmentRect
A pointer to a rectangle in window coordinates that allows you to align the window to a rectangle within the window. Set this parameter to `NIL` to align using the bounds of the window.

alignmentProc
Points to a function that allows you to provide your own alignment behavior. Set this parameter to `NIL` to use the standard behavior.

Discussion
For a specification of your alignment function, see `ICMAlignmentProc`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
BurntTextSampleCode
MakeEffectMovie
MovieGWorlds
QTCarbonShell
SimpleVideoOut

Declared In
`ImageCompression.h`

CDSequenceBusy

Checks the status of an asynchronous compression or decompression operation.

```
OSErr CDSequenceBusy (
   ImageSequence seqID
);
```

Parameters

seqID

Contains the unique sequence identifier that was returned by `DecompressSequenceBegin` (page 621) or `CompressSequenceBegin` (page 609).

Return Value

If there is no asynchronous operation in progress, `CDSequenceBusy` returns a 0 result code. If there is an asynchronous operation in progress, the result code is 1. Negative result codes indicate an error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

CDSequenceChangedSourceData

Notifies the compressor that the image source data has changed.

```
OSErr CDSequenceChangedSourceData (
   ImageSequenceDataSource sourceID
);
```

Parameters

sourceID

Contains the source identifier of the data source.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Use this function to indicate that the image has changed but the data pointer to that image has not changed. For example, if the data pointer points to the base address of a `PixMap` structure, the image in the `PixMap` can change, but the data pointer remains constant.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

CDSequenceDisposeDataSource

Disposes of a data source.

```
OSErr CDSequenceDisposeDataSource (
   ImageSequenceDataSource sourceID
);
```

Parameters

sourceID

The data source identifier that was returned by the CDSequenceNewDataSource (page 595) function.

Return Value

See Error Codes. Returns noErr if there is no error.

Discussion

Use this function to dispose of a data source created by the CDSequenceNewDataSource (page 595) function. All data sources are automatically disposed when the sequence they are associated with is disposed.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

ImageCompression.h

CDSequenceDisposeMemory

Disposes of memory allocated by the codec.

```
OSErr CDSequenceDisposeMemory (
   ImageSequence seqID,
   Ptr data
);
```

Parameters

seqID

Contains the unique sequence identifier that was returned by the DecompressSequenceBegin (page 621) function.

data

Points to the previously allocated memory block.

Return Value

See Error Codes. Returns noErr if there is no error.

Discussion

You call this function to release memory allocated by the CDSequenceNewMemory (page 597) function.

Special Considerations

Do not call CDSequenceDisposeMemory at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

CDSequenceEnd

Indicates the end of processing for an image sequence.

```
OSErr CDSequenceEnd (
   ImageSequence seqID
);
```

Parameters

seqID
Contains the unique sequence identifier that was returned by `DecompressSequenceBegin` (page 621) or `CompressSequenceBegin` (page 609).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Desktop Sprites
qteffects
qteffects.win
SGDataProcSample
VideoProcessing

Declared In
`ImageCompression.h`

CDSequenceEquivalentImageDescription

Reports whether two image descriptions are the same.

```
OSErr CDSequenceEquivalentImageDescription (
   ImageSequence seqID,
   ImageDescriptionHandle newDesc,
   Boolean *equivalent
);
```

Parameters

seqID
Contains the unique sequence identifier that was returned by the `DecompressSequenceBegin` (page 621) function.

newDesc

A handle to the `ImageDescription` structure structure that describes the compressed image.

equivalent

A pointer to a Boolean value. If the `ImageDescriptionHandle` provided in the `newDesc` parameter is equivalent to the `ImageDescription` structure currently in use by the image sequence, this value is set to TRUE. If the `ImageDescriptionHandle` is not equivalent, and therefore a new image sequence must be created to display an image using the new image description, this value is set to FALSE.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function allows an application to ask whether two image descriptions are the same. If they are, the decompressor does not have to create a new image decompression sequence to display those images.

Special Considerations

The Image Compression Manager can only implement part of this function by itself. There are some fields in the `ImageDescription` structure that it knows are irrelevant to the decompressor. If the Image Compression Manager determines that there are differences in fields that may be significant to the codec, it calls the function `ImageCodecIsImageDescriptionEquivalent` to ask the codec.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qteffects.win

qtsprites.win

qtwiredactions

qtwiredsprites

qtwiredspritesjr

Declared In

`ImageCompression.h`

CDSequenceEquivalentImageDescriptionS

Undocumented

```
OSErr CDSequenceEquivalentImageDescriptionS (
   ImageSequence seqID,
   ImageDescriptionHandle newDesc,
   Boolean *equivalent,
   Boolean *canSwitch
);
```

Parameters

seqID

Contains the unique sequence identifier that was returned by the `DecompressSequenceBegin` (page 621) function.

newDesc

A handle to the `ImageDescription` structure structure that describes the compressed image.

equivalent

A pointer to a Boolean value. If the `ImageDescriptionHandle` provided in the `newDesc` parameter is equivalent to the `ImageDescription` structure currently in use by the image sequence, this value is set to TRUE. If the `ImageDescriptionHandle` is not equivalent, and therefore a new image sequence must be created to display an image using the new image description, this value is set to FALSE.

canSwitch

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

CDSequenceFlush

Stops a decompression sequence, aborting processing of any queued frames.

```
OSErr CDSequenceFlush (
   ImageSequence seqID
);
```

Parameters

seqID

Contains the unique sequence identifier that was returned by `DecompressSequenceBegin` (page 621).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function is used to tell a decompressor component to stop processing of any queued scheduled asynchronous decompression. This is useful when several frames have been queued for decompression in the future and the application needs to suspend playback of the sequence.

For any outstanding frames, your application's completion routine, passed to `DecompressSequenceFrameWhen` (page 626), will be called with an error result of -1, indicating that the frame was cancelled. If any frames are currently being decompressed and cannot be cancelled, `CDSequenceFlush` waits until the frame has finished decompressing before returning.

Version Notes

Introduced in QuickTime 2.0.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

CDSequenceGetDataSource

Gets a data source for a decompression sequence.

```
OSErr CDSequenceGetDataSource (
   ImageSequence seqID,
   ImageSequenceDataSource *sourceID,
   OSType sourceType,
   long sourceInputNumber
);
```

Parameters

seqID
The image sequence that this source is associated with.

sourceID
A pointer to the source reference identifying this source.

sourceType
A four-character code describing how the input will be used. This value is passed by `CDSequenceNewDataSource` (page 595) when the source is created.

sourceInputNumber
A value passed by `CDSequenceNewDataSource` (page 595) when the source is created.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

CDSequenceInvalidate

Notifies the Image Compression Manager that the destination port for the given image decompression sequence has been invalidated.

```
OSErr CDSequenceInvalidate (
   ImageSequence seqID,
   RgnHandle invalRgn
);
```

Parameters

seqID
Contains the unique sequence identifier that was returned by `DecompressSequenceBegin` (page 621).

invalRgn

A handle to the region specifying the invalid portion of the image.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

You call this function to force the Image Compression Manager to redraw the screen bits on the next decompression operation.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

CDSequenceNewDataSource

Creates a new data source.

```
OSErr CDSequenceNewDataSource (
   ImageSequence seqID,
   ImageSequenceDataSource *sourceID,
   OSType sourceType,
   long sourceInputNumber,
   Handle dataDescription,
   ICMConvertDataFormatUPP transferProc,
   void *refCon
);
```

Parameters

seqID

The unique sequence identifier that was returned by the `DecompressSequenceBegin` (page 621) function.

sourceID

Returns the new data source identifier.

sourceType

A four-character code describing how the input will be used. This code is usually derived from the information returned by the codec. For example, if a mask plane was passed, this field might contain `'mask'`.

sourceInputNumber

More than one instance of a given source type may exist. The first occurrence should have a source input number of 1, the second a source input number of 2, and so on.

dataDescription

A handle to a data structure describing the input data. For compressed image data, this is an `ImageDescriptionHandle`.

transferProc

A routine that allows the application to transform the type of the input data to the kind of data preferred by the codec. The client of the codec passes the source data in the form most convenient for it. If the codec needs the data in another form, it can negotiate with the client or directly with the Image Compression Manager to obtain the required data format.

refCon

A reference constant to be passed to the transfer procedure. Use this parameter to point to a data structure containing any information your function needs.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function returns a `sourceID` parameter which must be passed to all other functions that reference the source. All data sources are automatically disposed when the sequence they are associated with is disposed.

```
// CDSequenceNewDataSource coding example
// See "Discovering QuickTime," page 309
{
    ImageSequenceDataSource     lSrc1 =0;
    // Store a description of the first GWorld in hImageDesc1
    nErr =MakeImageDescriptionForPixMap(GetGWorldPixMap(gWorld1),
                    &hImageDesc1);
    // Create a source from the GWorld description.
    nErr =CDSequenceNewDataSource(gEffectSequenceID,
                                    &lSrc1,
                                    'srcA',
                                    1,
                                    (Handle)hImageDesc1,
                                    NIL,
                                    0);
    // Set the data for source srcA to be the pixMap of gWorld1
    CDSequenceSetSourceData(lSrc1,
                            GetPixBaseAddr(GetGWorldPixMap(gWorld1)),
                            (**hImageDesc1).dataSize);
}
```

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qteffects

qteffects.win

qtshoweffect

VideoProcessing

vrscript.win

Declared In

`ImageCompression.h`

CDSequenceNewMemory

Requests codec-allocated memory.

```
OSErr CDSequenceNewMemory (
   ImageSequence seqID,
   Ptr *data,
   Size dataSize,
   long dataUse,
   ICMMemoryDisposedUPP memoryGoneProc,
   void *refCon
);
```

Parameters

seqID
: Contains the unique sequence identifier that was returned by the `DecompressSequenceBegin` (page 621) function.

data
: Returns a pointer to the allocated memory.

dataSize
: The requested size of the data buffer.

dataUse
: A code (see below) that indicates how the memory is to be used. For example, the memory may be used to store compressed image or mask plane data, or used as an offscreen image buffer. If there is no benefit to storing a particular kind of data in codec memory, the codec should deny the request for the memory allocation. See these constants:

memoryGoneProc
: A pointer to a callback function that will be called before disposing of the memory allocated by a codec, as described in `ICMMemoryDisposedProc`.

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

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Because many hardware decompression boards contain dedicated on-board memory, significant performance gains can be realized if this memory is used to store data before it is decompressed. When memory is allocated, a callback function must be provided, as described in `ICMMemoryDisposedProc`. The decompressor can dispose of all memory it has allocated at any time, but it calls the callback routine before disposing of the memory. A callback procedure is required because memory on the hardware decompression board may be limited. If the decompressor cannot deallocate memory as required, it is possible that an idle decompressor instance may be holding a large amount of memory, denying those resources to the currently active decompressor instance. When the callback procedure is called, the memory is still available. This allows any pending reads into the block to be canceled before the block is disposed. The decompressor disposing the memory must ensure that it is not disposing a block that it is currently using (that is, a block that contains the currently decompressing frame). To dispose of the memory, use `CDSequenceDisposeMemory` (page 590).

Special Considerations

Decompressor memory must never be disposed at interrupt time.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

CDSequenceSetSourceData

Sets data in a new frame to a specific data source.

```
OSErr CDSequenceSetSourceData (
   ImageSequenceDataSource sourceID,
   void *data,
   long dataSize
);
```

Parameters

sourceID
    Contains the source identifier of the data source.

data
    Points to the data. This pointer must contain a 32-bit clean address.

dataSize
    The size of the data buffer.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function is called to set data in a new frame to a specific source. For example, as a new frame of compressed data arrives at a source, `CDSequenceSetSourceData` will be called.

```
// CDSequenceSetSourceData coding example
// See "Discovering QuickTime," page 309
{
    ImageSequenceDataSource     lSrc1 =0;
    // Store a description of the first GWorld in hImageDesc1
    nErr =MakeImageDescriptionForPixMap(GetGWorldPixMap(gWorld1),
                &hImageDesc1);
    // Create a source from the GWorld description.
    nErr =CDSequenceNewDataSource(gEffectSequenceID,
                                &lSrc1,
                                'srcA',
                                1,
                                (Handle)hImageDesc1,
                                NIL,
                                0);
    // Set the data for source srcA to be the pixMap of gWorld1
    CDSequenceSetSourceData(lSrc1,
                        GetPixBaseAddr(GetGWorldPixMap(gWorld1)),
                        (**hImageDesc1).dataSize);
}
```

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qteffects

qteffects.win

qtshoweffect

VideoProcessing

vrscript.win

Declared In
`ImageCompression.h`

CDSequenceSetSourceDataQueue

Sets a data queue as the source for a decompression sequence.

```
OSErr CDSequenceSetSourceDataQueue (
   ImageSequenceDataSource sourceID,
   QHdrPtr dataQueue
);
```

Parameters

`sourceID`
: Contains the source identifier of the data source.

`dataQueue`
: A pointer to a `QHdr` structure.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

CDSequenceSetTimeBase

Sets a time base for a decompression sequence.

```
OSErr CDSequenceSetTimeBase (
   ImageSequence seqID,
   void *base
);
```

Parameters

seqID

A unique sequence identifier that was returned by `CompressSequenceBegin` (page 609).

base

A pointer to the time base for this operation. Your application obtains this time base identifier from `NewTimeBase` (page 261).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

When you run a visual effect outside a movie, you must designate a time base that will be used when the effect is run. The following code illustrates this use of `CDSequenceSetTimeBase`:

```
// CDSequenceSetTimeBase coding example
// See "Discovering QuickTime," page 310
timeBase =NewTimeBase();
SetTimeBaseRate(timeBase, 0);
CDSequenceSetTimeBase(gEffectSequenceID, timeBase);
```

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qteffects

qteffects.win

qtshoweffect

VideoProcessing

vrscript.win

Declared In

`ImageCompression.h`

CodecManagerVersion

Determines the version of the installed Image Compression Manager.

```
OSErr CodecManagerVersion (
   long *version
);
```

Parameters

version

A pointer to a long integer that is to receive the version information. The Image Compression Manager returns its version number into this location. The version number is a long integer value.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function returns the version information as a long integer value. Use this function to retrieve the version number associated with the Image Compression Manager that is installed on a particular computer.

Special Considerations

The Image Compression Manager provides a number of functions that allow your application to obtain information about the facilities available for image compression or about compressed images. Your application may use some of these functions to select a specific compressor or decompressor for a given operation or to determine how much memory to allocate to receive a decompressed image. In addition, your application may use some of these functions to determine the capabilities of the components that are available on the user's computer system. You can then condition the options your program makes available to the user based on the user's system configuration.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

CompAdd

Undocumented

```
void CompAdd (
   wide *src,
   wide *dst
);
```

Parameters

src

Undocumented

dst

Undocumented

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

CompCompare

Undocumented

```
long CompCompare (
   const wide *a,
   const wide *minusb
);
```

Parameters

a
    Undocumented

minusb
    Undocumented

Return Value
Undocumented

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

CompDiv

Undocumented

```
long CompDiv (
   wide *numerator,
   long denominator,
   long *remainder
);
```

Parameters

numerator
    Undocumented

denominator
    Undocumented

remainder
    Undocumented

Return Value
Undocumented

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
SoftVDigX

Declared In
`ImageCompression.h`

CompFixMul

Undocumented

```
void CompFixMul (
   wide *compSrc,
   Fixed fixSrc,
   wide *compDst
);
```

Parameters

compSrc
Undocumented

fixSrc
Undocumented

compDst
Undocumented

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

CompMul

Undocumented

```
void CompMul (
   long src1,
   long src2,
   wide *dst
);
```

Parameters

src1
Undocumented

src2
Undocumented

dst
Undocumented

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

CompMulDiv

Undocumented

```
void CompMulDiv (
   wide *co,
   long mul,
   long divisor
);
```

Parameters

`co`
: *Undocumented*

`mul`
: *Undocumented*

`divisor`
: *Undocumented*

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

CompMulDivTrunc

Undocumented

```
void CompMulDivTrunc (
   wide *co,
   long mul,
   long divisor,
   long *remainder
);
```

Parameters

`co`
: *Undocumented*

`mul`
: *Undocumented*

`divisor`
: *Undocumented*

`remainder`
: *Undocumented*

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

CompNeg

Undocumented

```
void CompNeg (
   wide *dst
);
```

Parameters

`dst`
: *Undocumented*

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

CompressImage

Compresses a single-frame image that is currently stored as a pixel map structure.

```
OSErr CompressImage (
   PixMapHandle src,
   const Rect *srcRect,
   CodecQ quality,
   CodecType cType,
   ImageDescriptionHandle desc,
   Ptr data
);
```

Parameters

`src`
: A handle to the image to be compressed. The image must be stored in a pixel map structure.

`srcRect`
: A pointer to a rectangle defining the portion of the image to compress.

quality

A constant (see below) that defines the desired compressed image quality. See these constants:

- `codecMinQuality`
- `codecLowQuality`
- `codecNormalQuality`
- `codecHighQuality`
- `codecMaxQuality`
- `codecLosslessQuality`

cType

A compressor type; see `Codec Identifiers`.

desc

A handle that is to receive a formatted `ImageDescription` structure. The Image Compression Manager resizes this handle for the returned image description structure. Your application should store this image description with the compressed image data.

data

Points to a location to receive the compressed image data. It is your program's responsibility to make sure that this location can receive at least as much data as indicated by the `GetMaxCompressionSize` (page 670) function. The Image Compression Manager places the actual size of the compressed image into the `dataSize` field of the `ImageDescription` structure structure referred to by the `desc` parameter. This pointer must contain a 32-bit clean address.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The following code sample illustrates the process of compressing and decompressing a pixel map.

```
// CompressImage coding example
// See "Discovering QuickTime," page 286
PicHandle GetQTCompressedPict (PixMapHandle hpmImage)
{
    long                    lMaxCompressedSize =0;
    Handle                  hCompressedData =NIL;
    Ptr                     pCompressedData;
    ImageDescriptionHandle  hImageDesc =NIL;
    OSErr                   nErr;
    PicHandle               hpicPicture =NIL;
    Rect                    rectImage =(**hpmImage).bounds;
    CodecType               dwCodecType =kJPEGCodecType;
    CodecComponent          codec =(CodecComponent)anyCodec;
    CodecQ                  dwSpatialQuality =codecNormalQuality;
    short                   nDepth =0;          // let ICM choose depth
    nErr =GetMaxCompressionSize(hpmImage, &rectImage, nDepth,
                                        dwSpatialQuality,
                                        dwCodecType,
                                        (CompressorComponent)codec,
                                        &lMaxCompressedSize);
    if (nErr !=noErr)
        return NIL;

    hImageDesc =(ImageDescriptionHandle)NewHandle(4);
    hCompressedData =NewHandle(lMaxCompressedSize);
    if ((hCompressedData !=NIL) && (hImageDesc !=NIL)) {
        MoveHHi(hCompressedData);
        HLock(hCompressedData);
        pCompressedData =StripAddress(*hCompressedData);

        nErr =CompressImage(hpmImage,
                                &rectImage,
                                dwSpatialQuality,
                                dwCodecType,
                                hImageDesc,
                                pCompressedData);

        if (nErr ==noErr) {
            ClipRect(&rectImage);
            hpicPicture =OpenPicture(&rectImage);
            nErr =DecompressImage(pCompressedData,
                                hImageDesc,
                                hpmImage,
                                &rectImage,
                                &rectImage,
                                srcCopy,
                                NIL);
            ClosePicture();
        }
        if (theErr || (GetHandleSize((Handle)hpicPicture) ==
                                    sizeof(Picture))) {
            KillPicture(hpicPicture);
            hpicPicture =NIL;
        }
    }
    if (hImageDesc !=NIL)
        DisposeHandle((Handle)hImageDesc);
    if (hCompressedData !=NIL)
        DisposeHandle(hCompressedData);
    return hpicPicture;
}
```

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
AddFrameToMovie
CocoaCreateMovie
Fiendishthngs
qteffects.win
ThreadsImportMovie

Declared In
`ImageCompression.h`

CompressPicture

Compresses a single-frame image stored as a picture structure and places the result in another picture.

```
OSErr CompressPicture (
   PicHandle srcPicture,
   PicHandle dstPicture,
   CodecQ quality,
   CodecType cType
);
```

Parameters

srcPicture

A handle to the source image, stored as a picture.

dstPicture

A handle to the destination for the compressed image. The compressor resizes this handle for the result data.

quality

A constant (see below) that defines the desired compressed image quality. See these constants:

`codecMinQuality`

`codecLowQuality`

`codecNormalQuality`

`codecHighQuality`

`codecMaxQuality`

`codecLosslessQuality`

cType

You must set this parameter to a valid compressor identifier; see `Codec Identifiers`. If the value passed in is 0, or `'raw '`, and the source picture is compressed, the destination picture is created as an uncompressed picture and does not require QuickTime for its display.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function compresses only image data. Any other types of data in the picture, such as text, graphics primitives, and previously compressed images, are not modified in any way and are passed through to the destination picture. This function supports parameters governing image quality and compressor type. The compressor infers the other compression parameters from the image data in the source picture.

Special Considerations

If a picture with multiple pixel maps and other graphical objects is passed, the pixel maps will be compressed individually and the other graphic objects will not be affected. This function does not use the graphics port.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

JPEG File Interchange Format

Declared In

`ImageCompression.h`

CompressPictureFile

Compresses a single-frame image stored as a picture file and places the result in another picture file.

```
OSErr CompressPictureFile (
   short srcRefNum,
   short dstRefNum,
   CodecQ quality,
   CodecType cType
);
```

Parameters

srcRefNum

A file reference number for the source `'PICT'` file.

dstRefNum

A file reference number for the destination `'PICT'` file. Note that the compressor overwrites the contents of the file referred to by `dstRefNum`. You must open this file with write permission. The destination file can be the same as the source file specified by the `srcRefNum` parameter.

quality

A constant (see below) that defines the desired compressed image quality. See these constants:

`codecMinQuality`

`codecLowQuality`

`codecNormalQuality`

`codecHighQuality`

`codecMaxQuality`

`codecLosslessQuality`

cType

A compressor type. You must set this parameter to a valid compressor type constant; see `Codec Identifiers`. If the value passed in is 0, or `'raw-'`, and the source picture is compressed, the destination picture is created as an uncompressed picture and does not require QuickTime to be displayed.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function supports parameters governing image quality and compressor type. The compressor infers the other compression parameters from the image data in the source picture file.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

CompressSequenceBegin

Signals the beginning of the process of compressing a sequence of frames.

```
OSErr CompressSequenceBegin (
   ImageSequence *seqID,
   PixMapHandle src,
   PixMapHandle prev,
   const Rect *srcRect,
   const Rect *prevRect,
   short colorDepth,
   CodecType cType,
   CompressorComponent codec,
   CodecQ spatialQuality,
   CodecQ temporalQuality,
   long keyFrameRate,
   CTabHandle ctable,
   CodecFlags flags,
   ImageDescriptionHandle desc
);
```

Parameters

seqID

A pointer to a field to receive the unique identifier for this sequence. You must supply this identifier to all subsequent Image Compression Manager functions that relate to this sequence.

src

A handle to a pixel map that will contain the image to be compressed. The image must be stored in a pixel map structure.

prev

A handle to a pixel map that will contain a previous image. The compressor uses this buffer to store a previous image against which the current image (stored in the pixel map referred to by the `src` parameter) is compared when performing temporal compression. This pixel map must be created at the same depth and with the same color table as the source image. The compressor manages the contents of this pixel map based upon several considerations, such as the key frame rate and the degree of difference between compared images. If you want the compressor to allocate this pixel map or if you do not want to perform temporal compression (that is, you have set the value of the `temporalQuality` parameter to 0), set this parameter to `NIL`.

srcRect

A pointer to a rectangle defining the portion of the image to compress. The compressor applies this rectangle to the image stored in the buffer referred to by the `src` parameter.

prevRect

A pointer to a rectangle defining the portion of the previous image to use for temporal compression. The compressor uses this portion of the previous image as the basis of comparison with the current image. The compressor ignores this parameter if you have not provided a buffer for previous images. This rectangle must be the same size as the source rectangle, which is specified with the `srcRect` parameter.

colorDepth

The depth at which the sequence is likely to be viewed. Compressors may use this as an indication of the color or grayscale resolution of the compressed images. If you set this parameter to 0, the Image Compression Manager determines the appropriate value for the source image. Values of 1, 2, 4, 8, 16, 24, and 32 indicate the number of bits per pixel for color images. Values of 34, 36, and 40 indicate 2-bit, 4-bit, and 8-bit grayscale, respectively, for grayscale images. Your program can determine which depths are supported by a given compressor by examining the compressor information structure returned by the `GetCodecInfo` (page 652) function.

cType

You must set this parameter to a valid compressor type constant. See `Codec Identifiers`.

codec

Specify a particular compressor by setting this parameter to its compressor identifier. Alternatively, you may use a special identifier (see below). Specifying a component instance may be useful if you have previously set some parameter on a specific instance of a codec field and want to make sure that the specified instance is used for that operation. See these constants:

spatialQuality

A pointer to a field containing a constant (see below) that defines the desired compressed image quality. You can change the `value` of this parameter for an active sequence by calling `SetCSequenceQuality` (page 720). See these constants:

`codecMinQuality`

`codecLowQuality`

`codecNormalQuality`

`codecHighQuality`

`codecMaxQuality`

`codecLosslessQuality`

temporalQuality

A pointer to a field containing a constant (see below) that defines the desired temporal quality. This parameter governs the level of compression you desire with respect to information between successive frames in the sequence. Set to 0 if you do not want temporal compression. You can change the `value` of this parameter for an active sequence by calling `SetCSequenceQuality` (page 720).

keyFrameRate

Specifies the maximum number of frames allowed between key frames. The compressor determines the optimum placement for key frames based upon the amount of redundancy between adjacent images in the sequence. Consequently, the compressor may insert key frames more frequently than you have requested. However, the compressor never places fewer key frames than is indicated by the setting of the `keyFrameRate` parameter. The compressor ignores this parameter if you have not requested temporal compression (that is, you have set the `temporalQuality` parameter to 0). If you pass in 0 in this parameter, this indicates that there are no key frames in the sequence. If you pass in any other number, it specifies the number of non-key frames between key frames. Set this parameter to 1 to specify all key frames, to 2 to specify every other frame as a key frame, to 3 to specify every third frame as a key frame, and so forth. Your application may change the key frame rate for an active sequence by calling `SetCSequenceKeyFrameRate` (page 718).

ctable

A handle to a custom color lookup table. Your program may use this parameter to indicate a custom color lookup table to be used with this image. If the value of the `colorDepth` parameter is less than or equal to 8 and the custom color lookup table is different from that of the source pixel map (that is, the `ctSeed` field values differ in the two pixel maps), the compressor remaps the colors of the image to the custom colors. If you set the `colorDepth` parameter to 16, 24, or 32, the compressor stores the custom color table with the compressed image. The compressor may use the table to specify the best colors to use when displaying the image at lower bit depths. The compressor ignores the `ctable` parameter when `colorDepth` is set to 33, 34, 36, or 40. If you set this parameter to `NIL`, the compressor uses the color lookup table from the source pixel map.

`flags`

Contains flags (see below) that provide further control information. You must set either `codecFlagUpdatePrevious` or `codecFlagUpdatePreviousComp` to 1. Set unused flags to 0. See these constants:

`codecFlagUpdatePrevious`

`codecFlagUpdatePreviousComp`

`codecFlagWasCompressed`

`desc`

A handle that is to receive a formatted `ImageDescription` structure. The Image Compression Manager resizes this handle for the returned image description structure. Your application should store this image description with the compressed sequence. During the compression operation, the Image Compression Manager and the compressor component update the contents of this image description. Consequently, you should not store the image description until the sequence has been completely processed.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The Image Compression Manager prepares for a sequence-compression operation by reserving appropriate system resources. Hence you must call `CompressSequenceBegin` before you call `CompressSequenceFrame` (page 612).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Desktop Sprites

qteffects.win

qtsprites.win

qtwiredactions

qtwiredsprites

Declared In

`ImageCompression.h`

CompressSequenceFrame

Compresses one of a sequence of frames.

```
OSErr CompressSequenceFrame (
   ImageSequence seqID,
   PixMapHandle src,
   const Rect *srcRect,
   CodecFlags flags,
   Ptr data,
   long *dataSize,
   UInt8 *similarity,
   ICMCompletionProcRecordPtr asyncCompletionProc
);
```

Parameters

seqID

Unique sequence identifier that was returned by `CompressSequenceBegin` (page 609).

src

A handle to a pixel map that contains the image to be compressed. The image must be stored in a pixel map structure.

srcRect

A pointer to a rectangle defining the portion of the image to compress. The compressor applies this rectangle to the image stored in the buffer referred to by the `src` parameter.

flags

Specifies flags (see below) that provide further control information. You must set either `codecFlagUpdatePrevious` or `codecFlagUpdatePreviousComp` to 1. Set unused flags to 0. See these constants:

`codecFlagUpdatePrevious`

`codecFlagWasCompressed`

`codecFlagUpdatePreviousComp`

`codecFlagForceKeyFrame`

`codecFlagLiveGrab`

data

Points to a location to receive the compressed image data. It is your program's responsibility to make sure that this location can receive at least as much data as indicated by the `GetMaxCompressionSize` (page 670) function. The Image Compression Manager places the actual size of the compressed image into the field referred to by the `dataSize` parameter. This pointer must contain a 32-bit clean address.

dataSize

A pointer to a field that is to receive the size, in bytes, of the compressed image.

similarity

A pointer to a field that is to receive a similarity value. The `CompressSequenceFrame` function returns a value that indicates the similarity of the current frame to the previous frame. A value of 0 indicates that the current frame is a key frame in the sequence. A value of 255 indicates that the current frame is identical to the previous frame. Values from 1 through 254 indicate relative similarity, ranging from very different (1) to very similar (254).

asyncCompletionProc

Points to an `ICMCompletionProc` callback. The compressor calls your completion function when an asynchronous compression operation is complete. You can cause the compression to be performed asynchronously by specifying a completion function if the compressor supports asynchronous compression.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
The Image Compression Manager prepares for a sequence-compression operation by reserving appropriate system resources. Hence you must call `CompressSequenceBegin` (page 609) before you call `CompressSequenceFrame`.

Special Considerations

If you specify asynchronous operation, you must not read the compressed data until the compressor indicates that the operation is complete by calling your completion function. Set `asyncCompletionProc` to `NIL` to specify synchronous compression. If you set `asyncCompletionProc` to -1, the operation is performed asynchronously but the compressor does not call your completion function. If the `asyncCompletionProc` parameter is not `NIL`, the following conditions are in effect: the pixels in the source image must stay valid until the completion function is called with its `codecCompletionSource` flag, and the resulting compressed data is not valid until it is called with its `codecCompletionDest` flag set.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Desktop Sprites

qteffects.win

qtsprites.win

qtwiredactions

qtwiredsprites

Declared In
`ImageCompression.h`

CompShift

Undocumented

```
void CompShift (
   wide *src,
   short shift
);
```

Parameters

`src`

Undocumented

`shift`

Undocumented

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

CompSquareRoot

Undocumented

```
unsigned long CompSquareRoot (
   const wide *src
);
```

Parameters

`src`
: *Undocumented*

Return Value
Undocumented

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

CompSub

Undocumented

```
void CompSub (
   wide *src,
   wide *dst
);
```

Parameters

`src`
: *Undocumented*

`dst`
: *Undocumented*

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Dimmer2Effect

Dimmer2Effect.win

Declared In
`ImageCompression.h`

ConcatMatrix

Concatenates two matrices, combining the transformations described by both matrices into a single matrix.

```
void ConcatMatrix (
   const MatrixRecord *a,
   MatrixRecord *b
);
```

Parameters

a

A pointer to the source matrix.

b

A pointer to the destination matrix. The `ConcatMatrix` function performs a matrix multiplication operation on the two matrices and leaves the result in the matrix specified by this parameter.

Discussion

This is a matrix multiplication operation, as a result of which [B] =[B] x [A]. Note that matrix multiplication is not commutative.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

BurntTextSampleCode

Declared In

`ImageCompression.h`

ConvertImage

Converts the format of a compressed image.

```
OSErr ConvertImage (
   ImageDescriptionHandle srcDD,
   Ptr srcData,
   short colorDepth,
   CTabHandle ctable,
   CodecQ accuracy,
   CodecQ quality,
   CodecType cType,
   CodecComponent codec,
   ImageDescriptionHandle dstDD,
   Ptr dstData
);
```

Parameters

srcDD

A handle to the `ImageDescription` structure that describes the compressed image.

srcData

Points to the compressed image data. This pointer must contain a 32-bit clean address.

colorDepth

The depth at which the recompressed image is likely to be viewed. Decompressors may use this as an indication of the color or grayscale resolution of the compressed image. If you set this parameter to 0, the Image Compression Manager determines the appropriate value for the source image. Values of 1, 2, 4, 8, 16, 24, and 32 indicate the number of bits per pixel for color images. Values of 34, 36, and 40 indicate 2-bit, 4-bit, and 8-bit grayscale, respectively, for grayscale images. Your program can determine which depths are supported by a given compressor by examining the compressor information structure returned by the `GetCodecInfo` (page 652) function.

ctable

A handle to a custom color lookup table. Your program may use this parameter to indicate a custom color lookup table to be used with this image. If the value of the `colorDepth` parameter is less than or equal to 8 and the custom color lookup table is different from that of the source pixel map (that is, the `ctSeed` field values differ in the two pixel maps), the compressor remaps the colors of the image to the custom colors. If you set the `colorDepth` parameter to 16, 24, or 32, the compressor stores the custom color table with the compressed image. The compressor may use the table to specify the best colors to use when displaying the image at lower bit depths. The compressor ignores the `ctable` parameter when `colorDepth` is set to 33, 34, 36, or 40. If you set this parameter to `NIL`, the compressor uses the color lookup table from the source `ImageDescription` structure.

accuracy

A constant (see below) that defines the accuracy desired in the decompressed image. Values for this parameter are on the same scale as compression quality. For a good display of still images, you should specify at least `codecHighQuality`. See these constants:

`codecMinQuality`
`codecLowQuality`
`codecNormalQuality`
`codecHighQuality`
`codecMaxQuality`
`codecLosslessQuality`

quality

A constant (see below) that defines the desired compressed image quality.

cType

A compressor type; see `Codec Identifiers`.

codec

Specify a particular compressor by setting this parameter to its compressor identifier. Alternatively, you may use a special constant (see below). Specifying a component instance may be useful if you have previously set some parameter on a specific instance of a codec field and want to make sure that the specified instance is used for that operation. See these constants:

dstDD

A handle that is to receive a formatted `ImageDescription` structure. The Image Compression Manager resizes this handle for the returned `ImageDescription` structure. Your application should store this image description with the compressed image data.

dstData

Points to a location to receive the compressed image data. It is your program's responsibility to make sure that this location can receive at least as much data as indicated by `GetMaxCompressionSize` (page 670). The Image Compression Manager places the actual size of the compressed image into the `dataSize` field of the image description referred to by the `dstDD` parameter. This pointer must contain a 32-bit-clean address.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
The action of this function is essentially equivalent to decompressing and recompressing the image. During the decompression operation, the decompressor uses the `srcDD`, `srcData`, and accuracy parameters. During the subsequent compression operation, the compressor uses the `colorDepth`, `ctable`, `cType`, codec, quality, `dstDD`, and `dstData` parameters.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

CopyMatrix

Copies the contents of one matrix into another matrix.

```
void CopyMatrix (
   const MatrixRecord *m1,
   MatrixRecord *m2
);
```

Parameters

m1

The source matrix for the copy operation.

m2

A pointer to the destination matrix for the copy operation. `CopyMatrix` copies the values from the matrix specified by the m1 parameter into this matrix.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

CountImageDescriptionExtensionType

Counts the number of extensions of a given type in an ImageDescriptionHandle.

```
OSErr CountImageDescriptionExtensionType (
   ImageDescriptionHandle desc,
   long idType,
   long *count
);
```

Parameters

desc

A handle to the `ImageDescription` structure with the extensions to be counted.

idType

Indicates the type of extension to be counted in the specified `ImageDescription` structure. Set the `value` of this parameter to 0 to match any extension, and return a count of all of the extensions.

count

A pointer to an integer that indicates how many extensions of the given type are in the given `ImageDescription` structure.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

When used with `GetNextImageDescriptionExtensionType` (page 672), this function allows the application to determine the total set of extensions present in the `ImageDescription` structure designated by `desc`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

DecompressImage

Decompresses a single-frame image into a pixel map structure.

```
OSErr DecompressImage (
   Ptr data,
   ImageDescriptionHandle desc,
   PixMapHandle dst,
   const Rect *srcRect,
   const Rect *dstRect,
   short mode,
   RgnHandle mask
);
```

Parameters

data

Points to the compressed image data. This pointer must contain a 32-bit clean address.

desc

A handle to the `ImageDescription` structure that describes the compressed image.

dst

A handle to the pixel map where the decompressed image is to be displayed. Set the current graphics port to the port that contains this pixel map.

srcRect

A pointer to a rectangle defining the portion of the image to decompress. This rectangle must lie within the boundary rectangle of the compressed image, which is defined by `(0,0)` and `((**desc).width,(**desc).height)`. If you want to decompress the entire source image, set this parameter to `NIL`. If the parameter is `NIL`, the rectangle is set to the rectangle structure of the `ImageDescription` structure.

dstRect

A pointer to the rectangle into which the decompressed image is to be loaded. The compressor scales the source image to fit into this destination rectangle.

mode

The transfer mode for the operation, as listed in `Graphics Transfer Modes`.

mask

A handle to a clipping region in the destination coordinate system. If specified, the compressor applies this mask to the destination image. If you do not want to mask bits in the `destination`, set this parameter to `NIL`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Note that `DecompressImage` is invoked through the `StdPix` (page 732) function. The following code sample illustrates the process of compressing and decompressing a pixel map.

```
// DecompressImage coding example
// See "Discovering QuickTime," page 286
PicHandle GetQTCompressedPict (PixMapHandle hpmImage)
{
    long                    lMaxCompressedSize =0;
    Handle                  hCompressedData =NIL;
    Ptr                     pCompressedData;
    ImageDescriptionHandle  hImageDesc =NIL;
    OSErr                   nErr;
    PicHandle               hpicPicture =NIL;
    Rect                    rectImage =(**hpmImage).bounds;
    CodecType               dwCodecType =kJPEGCodecType;
    CodecComponent          codec =(CodecComponent)anyCodec;
    CodecQ                  dwSpatialQuality =codecNormalQuality;
    short                   nDepth =0;          // let ICM choose depth
    nErr =GetMaxCompressionSize(hpmImage, &rectImage, nDepth,
                                dwSpatialQuality,
                                dwCodecType,
                                (CompressorComponent)codec,
                                &lMaxCompressedSize);
    if (nErr !=noErr)
        return NIL;

    hImageDesc =(ImageDescriptionHandle)NewHandle(4);
    hCompressedData =NewHandle(lMaxCompressedSize);
    if ((hCompressedData !=NIL) && (hImageDesc !=NIL)) {
        MoveHHi(hCompressedData);
        HLock(hCompressedData);
        pCompressedData =StripAddress(*hCompressedData);
        nErr =CompressImage(hpmImage,
                                &rectImage,
                                dwSpatialQuality,
                                dwCodecType,
                                hImageDesc,
                                pCompressedData);

        if (nErr ==noErr) {
            ClipRect(&rectImage);
            hpicPicture =OpenPicture(&rectImage);
            nErr =DecompressImage(pCompressedData,
                                hImageDesc,
                                hpmImage,
                                &rectImage,
                                &rectImage,
                                srcCopy,
                                NIL);
            ClosePicture();
        }
        if (theErr || (GetHandleSize((Handle)hpicPicture) ==
                                    sizeof(Picture))) {
            KillPicture(hpicPicture);
            hpicPicture =NIL;
        }
    }
    if (hImageDesc !=NIL)
        DisposeHandle((Handle)hImageDesc);
    if (hCompressedData !=NIL)
        DisposeHandle(hCompressedData);
    return hpicPicture;
}
```

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
ColorMatching
Desktop Sprites
FastDitherUsingQT
qteffects.win
qtwiredactions

Declared In
`ImageCompression.h`

DecompressSequenceBegin

Obsolete. See DecompressSequenceBeginS.

```
OSErr DecompressSequenceBegin (
   ImageSequence *seqID,
   ImageDescriptionHandle desc,
   CGrafPtr port,
   GDHandle gdh,
   const Rect *srcRect,
   MatrixRecordPtr matrix,
   short mode,
   RgnHandle mask,
   CodecFlags flags,
   CodecQ accuracy,
   DecompressorComponent codec
);
```

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Cocoa - SGDataProc

DelegateOnlyComponent

ImproveYourImage

SGDataProcSample

VideoProcessing

Declared In
`ImageCompression.h`

DecompressSequenceBeginS

Sends a sample image to a decompressor.

```
OSErr DecompressSequenceBeginS (
   ImageSequence *seqID,
   ImageDescriptionHandle desc,
   Ptr data,
   long dataSize,
   CGrafPtr port,
   GDHandle gdh,
   const Rect *srcRect,
   MatrixRecordPtr matrix,
   short mode,
   RgnHandle mask,
   CodecFlags flags,
   CodecQ accuracy,
   DecompressorComponent codec
);
```

Parameters

seqID

A pointer to a field to receive the unique identifier for the sequence you are creating. You should use this identifier for subsequent calls relating to this decompression sequence.

desc

A handle to the `ImageDescription` structure that describes the compressed image.

data

Points to the compressed image data. This pointer must contain a 32-bit clean address. Ideally, you should pass a pointer to the first frame of the compressed image data, which lets the Image Compression Manager do a better job of preflighting the decompression sequence. If the image data is not available at the time of this call, you can pass `NIL` for this parameter and 0 for `dataSize`. If you pass `NIL` here, then your first call to `DecompressSequenceFrameWhen` (page 626) may require more setup time.

dataSize

The size of the data buffer, or 0 if you passed `NIL` in the `data` parameter.

port

Points to the `CGrafPort` structure for the destination image.

gdh

A handle to the `GDevice` structure for the destination image. You can pass `NIL` if the `GDevice` is implicit in the port selection (for example, if it is an offscreen graphics world).

srcRect

A pointer to a `Rect` structure that defines the portions of the image to decompress. Pass `NIL` if you want to decompress the entire source image. You can call `SetDSequenceSrcRect` (page 727) to change the source rectangle for an active decompression sequence.

matrix

Points to a `MatrixRecord` structure that specifies how to transform the image during decompression. Pass `NIL` to use the identity matrix. Your application can change the matrix for an active sequence by calling `SetDSequenceMatrix` (page 724).

mode

The transfer mode for the operation. See `Graphics Transfer Modes`. Your application can change the transfer mode for an active sequence by calling `SetDSequenceTransferMode` (page 728).

mask

A handle to a clipping region in the destination coordinate system. If specified, the compressor applies this mask to the destination image. If you do not want to mask bits in the `destination`, set this parameter to `NIL`. Your application can change the clipping mask for an active sequence by calling `SetDSequenceMask` (page 724).

flags

Buffer allocation flags (see below). See these constants:

`codecFlagUseScreenBuffer`

`codecFlagUseImageBuffer`

accuracy

A constant (see below) that defines the desired compression accuracy. See these constants:

`codecMinQuality`

`codecLowQuality`

`codecNormalQuality`

`codecHighQuality`

`codecMaxQuality`

`codecLosslessQuality`

codec

A decompressor identifier. Specify a particular decompressor by setting this parameter to its identifier. Alternatively, you may use a special identifier (see below). Specifying a component instance may be useful if you have previously set some parameter on a specific instance of a codec field and want to make sure that the specified instance is used for that operation. See these constants:

Return Value

See `Error Codes`. Returns `codecWouldOffscreenErr` if `codecFlagDontUseImageBuffer` is set and the codec requires an offscreen buffer to decompress to the destination port. Returns `noErr` if there is no error.

Discussion

This function lets you pass a compressed sample so a codec can perform preflighting before the first `DecompressSequenceFrameWhen` (page 626) call. To decompress a series of images, call it once to preflight the decompressor, make calls to `DecompressSequenceFrameWhen` to decompress each image in the sequence, then call `CDSequenceEnd` (page 591) when you are done.

Version Notes

Introduced in QuickTime 1.6.1.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

MungSaver

qteffects.win

qtshoweffect

VideoProcessing

vrscript.win

Declared In

`ImageCompression.h`

DecompressSequenceFrame

Obsolete. See DecompressSequenceFrameS.

```
OSErr DecompressSequenceFrame (
   ImageSequence seqID,
   Ptr data,
   CodecFlags inFlags,
   CodecFlags *outFlags,
   ICMCompletionProcRecordPtr asyncCompletionProc
);
```

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CompressMovies

ConvertToMovieJr

Inside Mac ICM Code

SGDataProcSample

VideoProcessing

Declared In

`ImageCompression.h`

DecompressSequenceFrameS

Queues a frame for decompression and specifies the size of the compressed data; new applications should use DecompressSequenceFrameWhen.

```
OSErr DecompressSequenceFrameS (
   ImageSequence seqID,
   Ptr data,
   long dataSize,
   CodecFlags inFlags,
   CodecFlags *outFlags,
   ICMCompletionProcRecordPtr asyncCompletionProc
);
```

Parameters

seqID

Contains the unique sequence identifier that was returned by the `DecompressSequenceBegin` (page 621) function.

data

Points to the compressed image data. This pointer must contain a 32-bit clean address.

dataSize

The size of the data buffer.

inFlags

Contains flags (see below) that provide further control information. See these constants:

`codecFlagNoScreenUpdate`

`codecFlagDontOffscreen`

`codecFlagOnlyScreenUpdate`

outFlags

Contains status flags (see below). The decompressor updates these flags at the end of the decompression operation. See these constants:

`codecFlagUsedNewImageBuffer`

`codecFlagUsedImageBuffer`

`codecFlagDontUseNewImageBuffer`

`codecFlagInterlaceUpdate`

`codecFlagCatchUpDiff`

asyncCompletionProc

Points to an `ICMCompletionProcRecord` structure. The compressor calls your completion function when an asynchronous decompression operation is complete. You can cause the decompression to be performed asynchronously by specifying a completion function. If you specify asynchronous operation, you must not read the decompressed image until the decompressor indicates that the operation is complete by calling your completion function. Set `asyncCompletionProc` to `NIL` to specify synchronous decompression. If you set `asyncCompletionProc` to -1, the operation is performed asynchronously but the decompressor does not call your completion function.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function accepts the same parameters as the `DecompressSequenceFrame` (page 624) function, with the addition of the `dataSize` parameter.

Special Considerations

New applications should use `DecompressSequenceFrameWhen`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Cocoa - SGDataProc

ImproveYourImage

MungSaver

SGDataProcSample

VideoProcessing

Declared In

`ImageCompression.h`

DecompressSequenceFrameWhen

Queues a frame for decompression and specifies the time at which decompression will begin.

```
OSErr DecompressSequenceFrameWhen (
   ImageSequence seqID,
   Ptr data,
   long dataSize,
   CodecFlags inFlags,
   CodecFlags *outFlags,
   ICMCompletionProcRecordPtr asyncCompletionProc,
   const ICMFrameTimeRecord *frameTime
);
```

Parameters

`seqID`

Contains the unique sequence identifier that was returned by the `DecompressSequenceBegin` (page 621) function.

`data`

Points to the compressed image data. This pointer must contain a 32-bit clean address.

`dataSize`

The size of the data buffer.

`inFlags`

Contains flags (see below) that provide further control information. See these constants:

`codecFlagNoScreenUpdate`

`codecFlagDontOffscreen`

`codecFlagOnlyScreenUpdate`

outFlags

Contains status flags (see below). The decompressor updates these flags at the end of the decompression operation. See these constants:

`codecFlagUsedNewImageBuffer`

`codecFlagUsedImageBuffer`

`codecFlagDontUseNewImageBuffer`

`codecFlagInterlaceUpdate`

`codecFlagCatchUpDiff`

asyncCompletionProc

Points to an `ICMCompletionProcRecord` structure. The compressor calls your completion function when an asynchronous decompression operation is complete. You can cause the decompression to be performed asynchronously by specifying a completion function. If you specify asynchronous operation, you must not read the decompressed image until the decompressor indicates that the operation is complete by calling your completion function. Set `asyncCompletionProc` to `NIL` to specify synchronous decompression. If you set `asyncCompletionProc` to -1, the operation is performed asynchronously but the decompressor does not call your completion function.

frameTime

Points to an `ICMFrameTimeRecord` structure, which contains the frame's time information, including the time at which the frame should be displayed, its duration, and the movie's playback rate. This parameter can be `NIL`, in which case the decompression operation will happen immediately.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The following code snippet shows this function being used to execute one frame of a visual effect.

```
// DecompressSequenceFrameWhen coding example
// See "Discovering QuickTime," page 310
// Decompress a single step of the effect sequence.
OSErr RunEffect(TimeValue lTime, int nNumberOfSteps)
{
    OSErr               nErr =noErr;
    ICMFrameTimeRecord  ftr;
    // Set the timebase time to the step of the sequence to be rendered
    SetTimeBaseValue(timeBase, lTime, nNumberOfSteps);
    ftr.value.lo            =lTime;
    ftr.value.hi            =0;
    ftr.scale               =nNumberOfSteps;
    ftr.base                =0;
    ftr.duration            =nNumberOfSteps;
    ftr.rate                =0;
    ftr.recordSize          =sizeof(ftr);
    ftr.frameNumber         =1;
    ftr.flags               =icmFrameTimeHasVirtualStartTimeAndDuration;
    ftr.virtualStartTime.lo     =0;
    ftr.virtualStartTime.hi     =0;
    ftr.virtualDuration         =nNumberOfSteps;
    HLock(hEffectDesc);
    DecompressSequenceFrameWhen(gEffectSequenceID,
                            StripAddress(*hEffectDesc),
                            GetHandleSize(hEffectDesc),
                            0,
                            0,
                                NIL,
                                &ftr);
    HUnlock(hEffectDesc);
}
```

Special Considerations

If the current decompressor component does not support scheduled asynchronous decompression, the Image Compression Manager returns an error code of `codecCantWhenErr`. In this case, the application will need to reissue the request with the `frameTime` parameter set to `NIL`. If the decompressor cannot service your request at a particular time (for example, if its queue is full), the Image Compression Manager returns an error code of `codecCantQueueErr`. The best way to determine whether a decompressor component supports this function is to call the function and test the result code. A decompressor's ability to honor the request may change based on screen depth, clipping settings, and so on.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qteffects

qteffects.win

qtshoweffect

VideoProcessing

vrscript.win

Declared In

`ImageCompression.h`

DisposeCodecNameList

Disposes of the compressor name list structure you obtained by calling GetCodecNameList.

```
OSErr DisposeCodecNameList (
   CodecNameSpecListPtr list
);
```

Parameters

`list`

Points to the compressor name list to be disposed of. You obtain the compressor list by calling `GetCodecNameList` (page 652).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

DragAlignedGrayRgn

Drags a specified gray region along an optimal alignment grid.

```
long DragAlignedGrayRgn (
   RgnHandle theRgn,
   Point startPt,
   Rect *boundsRect,
   Rect *slopRect,
   short axis,
   UniversalProcPtr actionProc,
   Rect *alignmentRect,
   ICMAlignmentProcRecordPtr alignmentProc
);
```

Parameters

theRgn

A handle to the specified region for this operation. When the user holds down the mouse button, `DragAlignedGrayRgn` pulls a gray outline of the region around following the movement of the mouse until the mouse button is released.

startPt

The point where the mouse button was originally pressed in the local coordinates of the current graphics port.

boundsRect

A pointer to the boundary rectangle of the current graphics port. The offset point follows the mouse location except that `DragAlignedGrayRgn` never moves the offset point outside this rectangle. This limits the travel of the region's outline, not the movements of the mouse.

slopRect

A pointer to the slop rectangle that completely encloses the boundary rectangle so that the user is allowed some flexibility in moving the mouse.

axis

Allows you to constrain the region's motion to only one axis (see constants below). See these constants:

actionProc

Points to a function that defines some action to be performed repeatedly as long as the user holds down the mouse button. The function should have no parameters. If the `actionProc` parameter is `NIL`, `DragAlignedGrayRgn` simply retains control until the mouse button is released.

alignmentRect

A pointer to a rectangle within the bounds of the region specified in the parameter `theRgn`. Pass `NIL` to align using the `bounds` of the parameter `theRgn`.

alignmentProc

A pointer to your alignment behavior function; see `ICMAlignmentProc`. Pass `NIL` to use the standard behavior.

Return Value

The difference between the point where the mouse button was pressed and the offset point; that is, the point in the region whose horizontal and vertical offsets from the upper-left corner of the region's enclosing rectangle are the same as the offsets of the starting point when the user pressed the mouse button. The vertical difference between the starting point and the offset point is stored in the high-order word of the return value and the horizontal difference is stored in the low-order word.

Discussion
This function limits the movement of the region defined by `theRgn` according to the constraints set by the `boundsRect` and `slopRect` parameters. While the cursor is inside the `boundsRect` rectangle, the region's outline follows it normally. If the mouse button is released while the cursor is within this rectangle, the return value reflects the simple distance that the cursor moved in each dimension. When the cursor moves outside the `boundsRect` rectangle, the offset point stops at the edge of the `boundsRect` rectangle. If the mouse button is released while the cursor is outside the `boundsRect` rectangle but inside the `slopRect` rectangle, the return value reflects only the difference between the starting point and the offset point, regardless of how far outside of the `boundsRect` rectangle the cursor may have moved. (Note that part of the region can fall outside the `boundsRect` rectangle, but not the offset point.) When the cursor moves outside the `slopRect` rectangle, the region's outline disappears from the screen. `DragAlignedGrayRgn` continues to track the cursor, however, and if the cursor moves back into the `slopRect` rectangle, the outline reappears. If the mouse button is released while the cursor is anywhere inside the `slopRect` rectangle, the window is redrawn in its new location, calculated from the value returned by `DragAlignedGrayRgn` If the mouse button is released while the cursor is outside the `slopRect` rectangle, both words of the return value are set to 0x8000 and the window does not move from its original location.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

DragAlignedWindow

Drags the specified window along an optimal alignment grid.

```
void DragAlignedWindow (
   WindowRef wp,
   Point startPt,
   Rect *boundsRect,
   Rect *alignmentRect,
   ICMAlignmentProcRecordPtr alignmentProc
);
```

Parameters

`wp`
A window pointer to the window to be dragged.

`startPt`
A point that is equal to the point where the mouse button was pressed (in global coordinates, as stored in the `where` field of the event structure). `DragAlignedWindow` pulls a gray outline of the window around the screen, following the movements of the mouse until the button is released.

`boundsRect`
Points to the boundary rectangle in global coordinates. If the mouse button is released when the mouse position is outside the limits of the boundary rectangle, `DragAlignedWindow` returns without moving the window or making it the active window. For a document window, the boundary rectangle typically is four pixels in from the menu bar and from the other edges of the screen, to ensure that there won't be less than a four-pixel-square area of the title bar visible on the screen.

alignmentRect

Points to a rectangle in window coordinates that allows you to align the window to a rectangle within the window. Set this parameter to `NIL` to align using the bounds of the window.

alignmentProc

A pointer to your alignment behavior function; see `ICMAlignmentProc`. Pass `NIL` to use the standard behavior.

Discussion

The following code sample illustrates the use of `DragAlignedWindow`:

```
// DragAlignedWindow coding example
// See "Discovering QuickTime," page 265
Boolean IsQuickTimeInstalled (void)
{
    OSErr       nErr;
    long        lResult;
    nErr =Gestalt(gestaltQuickTime, &lResult);
    return (nErr ==noErr);
}
void MyInitialize (void)
{
    InitGraf(&qd.thePort);
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs(NIL);
    MaxApplZone();
    EnterMovies();
}
WindowRef MakeMyWindow (void)
{
    WindowRef   pMacWnd;
    Rect        rectWnd ={0, 0, 120, 160};
    Rect        rectBest;
    // figure out the best monitor for the window
    GetBestDeviceRect(NIL, &rectBest);
    // put the window in the top left corner of that monitor
    OffsetRect(&rectWnd, rectBest.left + 10, rectBest.top + 50);
    // create the window
    pMacWnd =NewCWindow(NIL, &rectWnd, "\pGrabber",
                        TRUE, noGrowDocProc, (WindowRef)-1,
                        TRUE, 0);
    // set the port to the new window
    SetPort(pMacWnd);
    return pMacWnd;
}
main (void)
{
    WindowRef           pMacWnd;
    SeqGrabComponent    seqGrab;
    SGChannel           sgchanVideo, sgchanSound;
    Boolean             bDone =FALSE;
    OSErr               nErr;
    MyInitialize();
    pMacWnd =MakeMyWindow();
    seqGrab =MakeMySequenceGrabber(pMacWnd);
    if (seqGrab ==NIL)
```

```
        return;
    MakeMyGrabChannels(seqGrab, &sgchanVideo, &sgchanSound,
                        &pMacWnd->
portRect, FALSE);
    nErr =SGStartPreview(seqGrab);
    while (!bDone) {
        ICMAlignmentProcRecord  apr;
        short                   nPart;
        WindowRef               pWhichWnd;
        EventRecord             er;
        GetNextEvent(everyEvent, &er);
        switch (er.what) {
            case nullEvent:     // give the sequence grabber time
                nErr =SGIdle(seqGrab);
                if (nErr !=noErr)
                    bDone =TRUE;
                break;
            case updateEvt:
                if (er.message ==(long)pMacWnd) {
                    // inform the sequence grabber of the update
                    SGUpdate(seqGrab,((WindowPeek)
                                     pMacWnd)->
updateRgn);
                    // and swallow the update event
                    BeginUpdate(pMacWnd);
                    EndUpdate(pMacWnd);
                }
                break;
            case mouseDown:
                nPart =FindWindow(er.where, &pWhichWnd);
                if (pWhichWnd !=pMacWnd)
                    break;
                switch (nPart) {
                    case inContent:
                        // pause until mouse button is released
                        SGPause(seqGrab, TRUE);
                        while (StillDown())
                            SGPause(seqGrab, FALSE);
                            break;
                    case inGoAway:
                        bDone =TrackGoAway(pMacWnd, er.where);
                        break;
                    case inDrag:
                        // pause when dragging window so video
                        // doesn't draw in the wrong place
                        SGPause(seqGrab, TRUE);
                        SGGetAlignmentProc(seqGrab, &apr);
                        DragAlignedWindow(pMacWnd,
                                          er.where,
                                          &screenBits.bounds,
                                          NIL, &alignProc);
                        SGPause(seqGrab, FALSE);
                        break;
                }
                break;
        }
    }
    // clean up
    SGStop(seqGrab);
    CloseComponent(seqGrab);
    DisposeWindow(pMacWnd);
}
```

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
MakeEffectMovie
makeeffectslideshow
qtcontroller
qtwiredactions
SGDataProcSample

Declared In
`ImageCompression.h`

DrawPictureFile

Draws an image from a specified picture file in the current graphics port.

```
OSErr DrawPictureFile (
   short refNum,
   const Rect *frame,
   ICMProgressProcRecordPtr progressProc
);
```

Parameters

refNum
A file reference number for the source PICT file.

frame
A pointer to the rectangle into which the image is to be loaded. The compressor scales the source image to fit into this destination rectangle.

progressProc
Points to an `ICMProgressProc` callback. During the operation, the draw function may occasionally call a function you provide in order to report its progress; see `ICMProgressProcRecord`. If you have not provided a progress function, set this parameter to `NIL`. If you pass a value of -1, QuickTime provides a standard progress function.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function draws the picture that it finds in the picture file specified by the `refNum` parameter within the rectangle specified by the `frame` parameter. The Image Compression Manager performs any spooling that may be necessary when reading the picture file. Specify a clipping region appropriate for your picture before drawing it. If the clipping region is very large (as it is when a graphics port is initialized) and you want to scale the picture, the clipping region can become invalid when `DrawPictureFile` scales the clipping region,

in which case your picture will not be drawn. On the other hand, if the graphics port specifies a small clipping region, part of your drawing may be clipped when `DrawPictureFile` draws it. Setting a clipping region equal to the port rectangle of the current graphics port always sets a valid clipping region.

Special Considerations

When it scales fonts, `DrawPictureFile` changes the size of the font instead of scaling the bits. However, the widths used by `bitmap` fonts are not always linear. For example, the 12-point width isn't exactly 1/2 of the 24-point width. This can cause lines of text to become slightly longer or shorter as the picture is scaled. The easiest way to avoid such problems is to specify a destination rectangle that is the same size as the bounding rectangle for the picture.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
JPEG File Interchange Format

Declared In
`ImageCompression.h`

DrawTrimmedPicture

Draws an image that is stored as a picture into the current graphics port and trims that image to fit a specified region.

```
OSErr DrawTrimmedPicture (
   PicHandle srcPicture,
   const Rect *frame,
   RgnHandle trimMask,
   short doDither,
   ICMProgressProcRecordPtr progressProc
);
```

Parameters

`srcPicture`
A handle to the source image, stored as a picture.

`frame`
A pointer to the rectangle into which the decompressed image is to be loaded.

`trimMask`
A handle to a clipping region in the destination coordinate system. The decompressor applies this mask to the destination image and ignores any image data that fall outside the specified region. Set this parameter to `NIL` if you do not want to clip the source image.

`doDither`
Indicates whether to dither the image. Use this parameter if you want the image to be dithered when it is displayed on a lower-resolution screen (see below). See these constants:
`defaultDither`
`forceDither`
`suppressDither`

progressProc
: A pointer to an `ICMProgressProc` callback. During the compression operation, the compressor may occasionally call a function you provide in order to report its progress. If you have not provided a progress function, set this parameter to `NIL`. If you pass a value of -1, QuickTime provides a standard progress function.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function works with compressed image data; the source data stays compressed. The function trims the image to fit the specified clipping region. Note that if you just use a clip while making a picture, the data (though not visible) is still stored in the picture.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
DrawTextCodec

Declared In
`ImageCompression.h`

DrawTrimmedPictureFile

Draws an image that is stored as a picture file into the current graphics port and trims that image to fit a specified region.

```
OSErr DrawTrimmedPictureFile (
   short srcRefnum,
   const Rect *frame,
   RgnHandle trimMask,
   short doDither,
   ICMProgressProcRecordPtr progressProc
);
```

Parameters

srcRefnum
: A file reference number for the source PICT file.

frame
: A pointer to the rectangle into which the decompressed image is to be loaded.

trimMask
: A handle to a clipping region in the destination coordinate system. The decompressor applies this mask to the destination image and ignores any image data that fall outside the specified region. Set this parameter to `NIL` if you do not want to clip the source image. In this case, this function acts like `DrawPictureFile` (page 633).

doDither

Indicates whether to dither the image. Use this parameter if you want the image to be dithered when it is displayed on a lower-resolution screen (see below). See these constants:

`defaultDither`

`forceDither`

`suppressDither`

progressProc

A pointer to an `ICMProgressProc` callback. During the compression operation, the compressor may occasionally call a function you provide in order to report its progress. If you have not provided a progress function, set this parameter to `NIL`. If you pass a value of -1, QuickTime provides a standard progress function.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

You can use this function to save part of a picture, since the image data that is not within the trim region is ignored and is not included in the destination picture file. All the remaining objects in the resulting object are clipped.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

DrawTextCodec

Declared In

`ImageCompression.h`

EqualMatrix

Compares two matrices and returns a result that indicates whether the matrices are equal.

```
Boolean EqualMatrix (
   const MatrixRecord *m1,
   const MatrixRecord *m2
);
```

Parameters

m1

A pointer to one matrix for the compare operation.

m2

A pointer to the other matrix for the compare operation.

Return Value

TRUE if the matrices are equal, FALSE otherwise.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qdmediahandler

qdmediahandler.win

Declared In
`ImageCompression.h`

FCompressImage

Compresses a single-frame image that is currently stored as a pixel map structure, with added control over the compression process.

```
OSErr FCompressImage (
   PixMapHandle src,
   const Rect *srcRect,
   short colorDepth,
   CodecQ quality,
   CodecType cType,
   CompressorComponent codec,
   CTabHandle ctable,
   CodecFlags flags,
   long bufferSize,
   ICMFlushProcRecordPtr flushProc,
   ICMProgressProcRecordPtr progressProc,
   ImageDescriptionHandle desc,
   Ptr data
);
```

Parameters

src
A handle to the image to be compressed. The image must be stored in a pixel map structure.

srcRect
A pointer to a rectangle defining the portion of the image to compress.

colorDepth
The depth at which the image is likely to be viewed. Compressors may use this as an indication of the color or grayscale resolution of the compressed image. If you set this parameter to 0, the Image Compression Manager determines the appropriate value for the source image. Values of 1, 2, 4, 8, 16, 24, and 32 indicate the number of bits per pixel for color images. Values of 34, 36, and 40 indicate 2-bit, 4-bit, and 8-bit grayscale, respectively, for grayscale images. Your program can determine which depths are supported by a given compressor by examining the compressor information structure returned by the `GetCodecInfo` (page 652) function.

quality

A constant (see below) that defines the desired compressed image quality. See these constants:

`codecMinQuality`

`codecLowQuality`

`codecNormalQuality`

`codecHighQuality`

`codecMaxQuality`

`codecLosslessQuality`

cType

A compressor type. You must set this parameter to a valid compressor type constant.

codec

A compressor identifier. Specify a particular compressor by setting this parameter to its compressor identifier. Alternatively, you may use a special identifier (see below). Specifying a component instance may be useful if you have previously set some parameter on a specific instance of a codec field and want to make sure that the specified instance is used for that operation. See these constants:

ctable

A handle to a custom color lookup table. Your program may use this parameter to indicate a custom color lookup table to be used with this image. If the value of the `colorDepth` parameter is less than or equal to 8 and the custom color lookup table is different from that of the source pixel map (that is, the `ctSeed` field values differ in the two pixel maps), the compressor remaps the colors of the image to the custom colors. If you set the `colorDepth` parameter to 16, 24, or 32, the compressor stores the custom color table with the compressed image. The compressor may use the table to specify the best colors to use when displaying the image at lower bit depths. The compressor ignores the `ctable` parameter when `colorDepth` is set to 33, 34, 36, or 40. If you set this parameter to `NIL`, the compressor uses the color lookup table from the source pixel map.

flags

Contains a flag (see below) that indicates whether or not the image was previously compressed. See these constants:

`codecFlagWasCompressed`

bufferSize

The size of the buffer to be used by the data-unloading function specified by the `flushProc` parameter. If you have not specified a data-unloading function, set this parameter to 0.

flushProc

Points to an `ICMDataProc` data-unloading callback. If there is not enough memory to store the compressed image, the compressor calls a function you provide that unloads some of the compressed data. If you have not provided a data-unloading callback, set this parameter to `NIL`. In this case, the compressor writes the entire compressed image into the memory location specified by the `data` parameter.

progressProc

Points to an `ICMProgressProc` progress callback. During the compression operation, the compressor may occasionally call a function you provide in order to report its progress. If you have not provided a progress callback, set this parameter to `NIL`. If you pass a value of -1, QuickTime provides a standard progress function.

desc

A handle that is to receive a formatted `ImageDescription` structure. The Image Compression Manager resizes this handle for the returned `ImageDescription` structure. Your application should store this image description with the compressed image data.

data

Points to a location to receive the compressed image data. It is your program's responsibility to make sure that this location can receive at least as much data as indicated by the `GetMaxCompressionSize` (page 670) function. If there is not sufficient memory to store the compressed image, you may choose to write the compressed data to mass storage during the compression operation. Use the `flushProc` parameter to identify your data-unloading function to the compressor. This pointer must contain a 32-bit clean address. The Image Compression Manager places the actual size of the compressed image into the `dataSize` field of the `ImageDescription` structure referenced by the `desc` parameter.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function acts like `CompressImage` (page 605), but gives your application additional control over the parameters that guide the compression operation.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

BurntTextSampleCode

qtgraphimp.win

qtreadwritejpeg.win

vrmakepano

VRMakePano Library

Declared In

`ImageCompression.h`

FCompressPicture

Compresses a single-frame image stored as a picture structure and places the result in another picture, with added control over the compression process.

```
OSErr FCompressPicture (
   PicHandle srcPicture,
   PicHandle dstPicture,
   short colorDepth,
   CTabHandle ctable,
   CodecQ quality,
   short doDither,
   short compressAgain,
   ICMProgressProcRecordPtr progressProc,
   CodecType cType,
   CompressorComponent codec
);
```

Parameters

srcPicture

A handle to the source image, stored as a picture.

dstPicture

A handle to the destination for the compressed image. The compressor resizes this handle for the result data.

colorDepth

The depth at which the image is to be compressed. If you set this parameter to 0, the Image Compression Manager determines the appropriate value for the source image. Values of 1, 2, 4, 8, 16, 24, and 32 indicate the number of bits per pixel for color images. Values of 34, 36, and 40 indicate 2-bit, 4-bit, and 8-bit grayscale, respectively, for grayscale images. Your program can determine which depths are supported by a given compressor by examining the compressor information structure returned by the `GetCodecInfo` (page 652) function.

ctable

A handle to a custom color lookup table. Your program may use this parameter to indicate a custom color lookup table to be used with this image. If the value of the `colorDepth` parameter is less than or equal to 8 and the custom color lookup table is different from that of the source pixel map (that is, the `ctSeed` field values differ in the two pixel maps), the compressor remaps the colors of the image to the custom colors. If you set the `colorDepth` parameter to 16, 24, or 32, the compressor stores the custom color table with the compressed image. The compressor may use the table to specify the best colors to use when displaying the image at lower bit depths. The compressor ignores the `ctable` parameter when `colorDepth` is set to 33, 34, 36, or 40. If you set this parameter to `NIL`, the compressor uses the color lookup table from the source pixel map.

quality

A constant that defines the desired compressed image quality. See these constants:

`codecMinQuality`

`codecLowQuality`

`codecNormalQuality`

`codecHighQuality`

`codecMaxQuality`

`codecLosslessQuality`

doDither

A constant (see below) that indicates whether to dither the image. Use this parameter to indicate whether you want the image to be dithered when it is displayed on a lower-resolution screen. See these constants:

`defaultDither`

`forceDither`

`suppressDither`

compressAgain

Indicates whether to recompress compressed image data in the `picture`. Use this parameter to control whether any compressed image data that is in the source picture should be decompressed and then recompressed using the current parameters. Set the `value` of this parameter to TRUE to recompress such data. Set the `value` of the parameter to FALSE to leave the data as it is. Note that recompressing the data may have undesirable side effects, including image quality degradation.

progressProc

Points to an `ICMProgressProc` callback. During the compression operation, the compressor may occasionally call a function you provide in order to report its progress. If you have not provided a progress callback, set this parameter to `NIL`. If you pass a value of -1, QuickTime provides a standard progress function.

cType

A compressor type. You must set this parameter to a valid compressor type constant; see `Codec Identifiers`. If the value passed in is 0, or `'raw '`, the resulting picture is not compressed and does not require QuickTime to be displayed.

codec

A compressor identifier. Specify a particular compressor by setting this parameter to its compressor identifier. Alternatively, you may use a special identifier (see below). See these constants:

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
If a picture with multiple pixel maps and other graphical objects is passed, the pixel maps will be compressed individually and the other graphic objects will not be affected. `FCompressPicture` compresses only image data. Any other types of data in the picture, such as text, graphics primitives, and previously compressed images, are not modified in any way and are passed through to the destination picture. This function supports parameters governing image quality, compressor type, image depth, custom color tables, and dithering.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

FCompressPictureFile

Compresses a single-frame image stored as a picture file and places the result in another picture file, with added control over the compression process.

```
OSErr FCompressPictureFile (
   short srcRefNum,
   short dstRefNum,
   short colorDepth,
   CTabHandle ctable,
   CodecQ quality,
   short doDither,
   short compressAgain,
   ICMProgressProcRecordPtr progressProc,
   CodecType cType,
   CompressorComponent codec
);
```

Parameters

srcRefNum

A file reference number for the source PICT file.

dstRefNum

A file reference number for the destination PICT file. Note that the compressor overwrites the contents of the file referred to by `dstRefNum`. You must open this file with write permissions. The destination file may be the same as the source file specified by the `srcRefNum` parameter.

colorDepth

The depth at which the image is to be compressed. If you set this parameter to 0, the Image Compression Manager determines the appropriate value for the source image. Values of 1, 2, 4, 8, 16, 24, and 32 indicate the number of bits per pixel for color images. Values of 34, 36, and 40 indicate 2-bit, 4-bit, and 8-bit grayscale, respectively, for grayscale images. Your program can determine which depths are supported by a given compressor by examining the compressor capability structure returned by the `GetCodecInfo` (page 652) function.

ctable

A handle to a custom color lookup table. Your program may use this parameter to indicate a custom color lookup table to be used with this image. If the value of the `colorDepth` parameter is less than or equal to 8 and the custom color lookup table is different from that of the source pixel map (that is, the `ctSeed` field values differ in the two pixel maps), the compressor remaps the colors of the image to the custom colors. If you set the `colorDepth` parameter to 16, 24, or 32, the compressor stores the custom color table with the compressed image. The compressor may use the table to specify the best colors to use when displaying the image at lower bit depths. The compressor ignores the `ctable` parameter when `colorDepth` is set to 33, 34, 36, or 40. If you set this parameter to `NIL`, the compressor uses the color lookup table from the source pixel map.

quality

A constant (see below) that defines the desired compressed image quality. See these constants:

`codecMinQuality`

`codecLowQuality`

`codecNormalQuality`

`codecHighQuality`

`codecMaxQuality`

`codecLosslessQuality`

doDither

Indicates whether to dither the image. Use this parameter to indicate whether you want the image to be dithered when it is displayed on a lower-resolution screen. The following constants are available: See these constants:

`defaultDither`

`forceDither`

`suppressDither`

compressAgain

Indicates whether to recompress compressed image data in the `picture`. Use this parameter to control whether any compressed image data that is in the source picture should be decompressed and then recompressed using the current parameters. Set the `value` of this parameter to TRUE to recompress such data. Set the `value` of this parameter to FALSE to leave the data as it is. Note that recompressing the data may have undesirable side effects, including image quality degradation.

progressProc

Points to an `ICMProgressProc` callback. During the compression operation, the compressor may occasionally call a function you provide in order to report its progress.

cType

A compressor type. You must set this parameter to a valid compressor type constant.

codec
: A compressor identifier. Specify a particular compressor by setting this parameter to its compressor identifier. Alternatively, you may use a special identifier (see below). See these constants:

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function compresses only image data. Any other types of data in the file, such as text, graphics primitives, and previously compressed images, are not modified in any way and are passed through to the destination picture file. This function supports parameters governing image quality, compressor type, image depth, custom color tables, and dithering.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

FDecompressImage

Decompresses a single-frame image into a pixel map structure, with added control over the decompression process.

```
OSErr FDecompressImage (
   Ptr data,
   ImageDescriptionHandle desc,
   PixMapHandle dst,
   const Rect *srcRect,
   MatrixRecordPtr matrix,
   short mode,
   RgnHandle mask,
   PixMapHandle matte,
   const Rect *matteRect,
   CodecQ accuracy,
   DecompressorComponent codec,
   long bufferSize,
   ICMDataProcRecordPtr dataProc,
   ICMProgressProcRecordPtr progressProc
);
```

Parameters

data
: Points to the compressed image data. If the entire compressed image cannot be stored at this location, your application may provide a data-loading function (see the discussion of the `dataProc` parameter to this function). This pointer must contain a 32-bit clean address.

desc
: A handle to the `ImageDescription` structure that describes the compressed image.

dst
: A handle to the pixel map where the decompressed image is to be displayed. Set the current graphics port to the port that contains this pixel map.

srcRect

A pointer to a rectangle defining the portion of the image to decompress. This rectangle must lie within the boundary rectangle of the compressed image, which is defined by `(0,0)` and `((**desc).width,(**desc).height)`. If you want to decompress the entire source image, set this parameter to `NIL`. If the parameter is `NIL`, the rectangle is set to the rectangle structure of the `ImageDescription` structure.

matrix

Points to a matrix structure that specifies how to transform the image during decompression. You can use the matrix structure to translate or scale the image during decompression. If you do not want to apply such effects, set the `matrix` parameter to `NIL`.

mode

The transfer mode for the operation; see `Graphics Transfer Modes`.

mask

A handle to a clipping region in the destination coordinate system. If specified, the decompressor applies this mask to the destination image. If you do not want to mask bits in the `destination`, set this parameter to `NIL`.

matte

A handle to a pixel map that contains a blend matte. You can use the blend matte to cause the decompressed image to be blended into the destination pixel map. The matte can be defined at any supported pixel depth; the matte depth need not correspond to the source or destination depths. However, the matte must be in the coordinate system of the source image. If you do not want to apply a blend matte, set this parameter to `NIL`.

matteRect

A pointer to a rectangle defining a portion of the blend matte to apply. If you do not want to use the entire matte referred to by the `matte` parameter, use this parameter to specify a rectangle within that matte. If specified, this rectangle must be the same size as the rectangle specified by the `srcRect` parameter. If you want to use the entire matte, or if you are not providing a blend matte, set this parameter to `NIL`.

accuracy

A constant (see below) that defines the desired compression accuracy. For a good display of still images, you should specify at least `codecHighQuality`. See these constants:

`codecMinQuality`

`codecLowQuality`

`codecNormalQuality`

`codecHighQuality`

`codecMaxQuality`

`codecLosslessQuality`

codec

A decompressor identifier. Specify a particular decompressor by setting this parameter to its identifier. Alternatively, you may use a special identifier (see below). Specifying a component instance may be useful if you have previously set some parameter on a specific instance of a codec field and want to make sure that the specified instance is used for that operation. See these constants:

bufferSize

The size of the buffer to be used by the data-loading function specified by the `dataProc` parameter. If you have not specified a data-loading function, set this parameter to 0.

dataProc

Points to an ICMDataProc data-loading callback. If there is not enough memory to store the compressed image, the compressor calls a function you provide that loads more compressed data. If you have not provided a data-unloading callback, set this parameter to NIL. In this case, the compressor expects that the entire compressed image is in the memory location specified by the data parameter.

progressProc

Points to an ICMProgressProc progress callback. During the compression operation, the compressor may occasionally call a function you provide in order to report its progress. If you have not provided a progress callback, set this parameter to NIL. If you pass a value of -1, QuickTime provides a standard progress function.

Return Value

See Error Codes. Returns noErr if there is no error.

Discussion

This function gives your application greater control over the parameters that guide the decompression operation. If you find that you do not need this level of control, use DecompressImage (page 619). Note that this function is invoked through the StdPix (page 732) function.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

JPEG File Interchange Format

qtreadwritejpeg

qtreadwritejpeg.win

VelEng Wavelet

Declared In

ImageCompression.h

FindCodec

Determines which of the installed compressors or decompressors has been chosen to field requests made by using one of the special compressor identifiers.

```
OSErr FindCodec (
   CodecType cType,
   CodecComponent specCodec,
   CompressorComponent *compressor,
   DecompressorComponent *decompressor
);
```

Parameters

cType

You must set this parameter to a valid compressor type constant; see Codec Identifiers.

specCodec

A special codec identifier value (see below). See these constants:

compressor

A pointer to a field to receive the identifier for the compressor component. The Image Compression Manager returns the identifier of the compressor that meets the special characteristics you specify in the `specCodec` parameter. Note that this identifier may differ from the `value` of the field referred to by the `decompressor` field. The Image Compression Manager sets this field to 0 if it cannot find a suitable compressor component. Set this parameter to `NIL` if you do not want this information.

decompressor

A pointer to a field to receive the identifier for the decompressor component. The Image Compression Manager returns the identifier of the decompressor that meets the special characteristics you specify in the `specCodec` parameter. Note that this identifier may differ from the `value` of the field referred to by the `compressor` field. The Image Compression Manager sets this field to 0 if it cannot find a suitable decompressor component. Set this parameter to `NIL` if you do not want this information.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Some Image Compression Manager functions allow you to specify a particular compressor component by its identifier. For example, you may use the `codec` parameter to `CompressSequenceBegin` (page 609) to specify a particular compressor to do the compression. The Image Compression Manager also supports several special identifiers (see `specCodec` Constants, above) that allow you to exert some control over the component for a given action without having to know its identifier.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

FixExp2

Undocumented

```
Fixed FixExp2 (
   Fixed src
);
```

Parameters

src

A fixed integer.

Return Value

Undocumented

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

FixLog2

Undocumented

```
Fixed FixLog2 (
   Fixed src
);
```

Parameters

`src`

A fixed integer.

Return Value
Undocumented

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

FixMulDiv

Undocumented

```
Fixed FixMulDiv (
   Fixed src,
   Fixed mul,
   Fixed divisor
);
```

Parameters

`src`

Undocumented

`mul`

Undocumented

`divisor`

Undocumented

Return Value
Undocumented

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

FixPow

Undocumented

```
Fixed FixPow (
   Fixed base,
   Fixed exp
);
```

Parameters

`base`
: *Undocumented*

`exp`
: *Undocumented*

Return Value
Undocumented

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

FracSinCos

Undocumented

```
Fract FracSinCos (
   Fixed degree,
   Fract *cosOut
);
```

Parameters

`degree`
: *Undocumented*

`cosOut`
: *Undocumented*

Return Value
Undocumented

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GDGetScale

Returns the current scale of the given screen graphics device.

```
OSErr GDGetScale (
   GDHandle gdh,
   Fixed *scale,
   short *flags
);
```

Parameters

gdh

A handle to a screen graphics device.

scale

Points to a fixed-point field to hold the scale result.

flags

Points to a short integer that returns the `status` parameter flags for the video driver. Currently, 0 is always returned in this field.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GDHasScale

Returns the closest possible scaling that a particular screen device can be set to in a given pixel depth.

```
OSErr GDHasScale (
   GDHandle gdh,
   short depth,
   Fixed *scale
);
```

Parameters

gdh

A handle to a screen graphics device.

depth

The pixel depth of the screen device.

scale
: Points to a fixed-point scale value. On input, this field should be set to the desired scale value. On output, this field will contain the closest scale available for the given depth. A scale of 0x10000 indicates normal size, 0x20000 indicates double size, and so on.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function returns scaling information for a particular screen device for a requested depth. This function allows you to query a screen device without actually changing it. For example, if you specify 0x20000 but the screen device does not support it, `GDHasScale` returns `noErr` and a scale of 0x10000. Because this function checks for a supported depth, your requested depth must be supported by the screen device.

Special Considerations

`GDHasScale` references the video driver through the graphics device structure.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GDSetScale

Sets a screen graphics device to a new scale.

```
OSErr GDSetScale (
   GDHandle gdh,
   Fixed scale,
   short flags
);
```

Parameters

gdh
: A handle to a screen graphics device.

scale
: A fixed-point scale value.

flags
: Points to a short integer. It returns the `status` parameter flags for the video driver. Currently, 0 is always returned in this field.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GetBestDeviceRect

Selects the deepest of all available graphics devices, while treating 16-bit and 32-bit screens as having equal depth.

```
OSErr GetBestDeviceRect (
   GDHandle *gdh,
   Rect *rp
);
```

Parameters

gdh

A pointer to the handle of the rectangle for the chosen device. If you do not need the information in this parameter returned, specify `NIL`.

rp

A pointer to the rectangle that is adjusted for the height of the menu bar if the device is the main device. If you do not need the information in this parameter returned, specify `NIL`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function does not center a rectangle on a device. Rather, it returns the rectangle for the best device. The following code sample illustrates its use:

```
// GetBestDeviceRect coding example
// See "Discovering QuickTime," page 265
WindowRef MakeMyWindow (void)
{
    WindowRef   pMacWnd;
    Rect        rectWnd ={0, 0, 120, 160};
    Rect        rectBest;
    // figure out the best monitor for the window
    GetBestDeviceRect(NIL, &rectBest);
    // put the window in the top left corner of that monitor
    OffsetRect(&rectWnd, rectBest.left + 10, rectBest.top + 50);
    // create the window
    pMacWnd =NewCWindow(NIL, &rectWnd, "\pGrabber",
                            TRUE, noGrowDocProc, (WindowRef)-1, TRUE, 0);
    // set the port to the new window
    SetPort(pMacWnd);
    return pMacWnd;
}
```

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

DigitizerShell

MovieBrowser
Sequence Grabbing
SGDataProcSample

Declared In
`ImageCompression.h`

GetCodecInfo

Returns information about a single compressor component.

```
ComponentResult ADD_IMAGECODEC_BASENAME() GetCodecInfo
```

Parameters

`info`
A pointer to a `CodecInfo` structure. `GetCodecInfo` returns detailed information about the appropriate compressor component in this structure.

`cType`
Set this parameter to a valid compressor type constant; see `Codec Identifiers`. If you want information about any compressor of the type specified by this parameter, set the `codec` parameter to 0. The Image Compression Manager then returns information about the first compressor it finds of the type you have specified.

`codec`
Set this parameter to the component identifier of the specific compressor for the request, or to 0 for any compressor. Component identifiers are available in the `CodecNameSpecList` structure returned by `GetCodecNameList` (page 652).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Fiendishthngs

Declared In
`ImageCompression.h`

GetCodecNameList

Retrieves a list of installed compressor components or types.

```
OSErr GetCodecNameList (
   CodecNameSpecListPtr *list,
   short showAll
);
```

Parameters

list

A pointer to a field that is to receive a pointer to a `CodecNameSpecList` structure. The Image Compression Manager creates the appropriate list and returns a pointer to that list in the field specified by this parameter.

showAll

A short integer that controls the contents of the `list`. Set this parameter to 1 to receive a list of the names of all installed compressor components; the returned list contains one entry for each installed compressor. Set this parameter to 0 to receive a list of the types of installed compressor components; the returned list contains one entry for each installed compressor type.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The `CodecType` data type defines a field in the `CodecNameSpec` structure that identifies the compression method employed by a given compressor component. See `Codec Identifiers`. Apple Computer's Developer Technical Support group assigns these values so that they remain unique. These values correspond, in turn, to text strings that can identify the compression method to the user.

Special Considerations

`GetCodecNameList` creates the `CodecNameSpecList` structure in your application's current heap zone.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GetCompressedImageSize

Determines the size, in bytes, of a compressed image.

```
ComponentResult ADD_IMAGECODEC_BASENAME() GetCompressedImageSize
```

Parameters

desc

A handle to the `ImageDescription` structure that defines the compressed image for the operation.

data

Points to the compressed image data. This pointer must contain a 32-bit clean address.

bufferSize

The size of the buffer to be used by the data-loading function specified by the `dataProc` parameter. If you have not specified a data-loading function, set this parameter to 0.

dataProc

Points to an `ICMDataProc` callback. If the data stream is not all in memory when your program calls `GetCompressedImageSize`, the compressor calls a function you provide that loads more compressed data. If you have not provided a data-loading callback, set this parameter to `NIL`. In this case, the entire image must be in memory at the location specified by the `data` parameter.

dataSize

A pointer to a field that is to receive the size, in bytes, of the compressed image.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Most applications do not need to use this function because compressed images have a corresponding `ImageDescription` structure with a size field. You only need to use this function if you do not have an image description structure associated with your data; for example, when you are taking a compressed image out of a movie one frame at a time.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GetCompressedPixMapInfo

Retrieves information about a compressed image.

```
OSErr GetCompressedPixMapInfo (
   PixMapPtr pix,
   ImageDescriptionHandle *desc,
   Ptr *data,
   long *bufferSize,
   ICMDataProcRecord *dataProc,
   ICMProgressProcRecord *progressProc
);
```

Parameters

pix

Points to a structure that holds encoded compressed image data.

desc

A pointer to a field that is to receive a handle to the `ImageDescription` structure that defines the compressed image. If you are not interested in this information, specify `NIL` in this parameter.

data

A pointer to a field that is to receive a pointer to the compressed image data. If the entire compressed image cannot be stored at this location, you can define a data-loading function for this operation. If you are not interested in this information, you may specify `NIL` in this parameter.

bufferSize

A pointer to a field that is to receive the size of the buffer to be used by the data-loading function specified by the `dataProc` parameter. If there is no data-loading function defined for this operation, this parameter is ignored. If you are not interested in this information, you may specify `NIL` in this parameter.

dataProc

A pointer to an `ICMDataProc` callback. If there is not enough memory to store the compressed image, the decompressor calls a function you provide that loads more compressed data. If there is no data-loading function for this image, the function sets the `dataProc` field in the function structure to `NIL`. If you are not interested in this information, specify `NIL` in this parameter.

progressProc

A pointer to an `ICMProgressProc` callback. During a decompression operation, the decompressor may occasionally call a function you provide in order to report its progress. If there is no progress function for this image, the function sets the `progressProc` field in the function structure to `NIL`. If you pass a value of -1, QuickTime provides a standard progress function. If you are not interested in progress information, specify `NIL` in this parameter.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Desktop Sprites

DesktopSprites

DesktopSprites.win

JPEG File Interchange Format

WiredSprites

Declared In

`ImageCompression.h`

GetCompressionTime

Determines the estimated amount of time required to compress a given image.

```
ComponentResult ADD_IMAGECODEC_BASENAME() GetCompressionTime
```

Parameters

src

A handle to the source image. The source image must be stored in a pixel map structure. The compressor uses only the bit depth of this image to determine the compression time. You may set this parameter to `NIL` if you are interested only in information about quality settings.

srcRect

A pointer to a rectangle defining the portion of the source image to compress. You may set this parameter to `NIL` if you are interested only in information about quality settings. `GetCompressionTime` then uses the bounds of the source pixel map.

colorDepth

The depth at which the image is to be compressed. If you set this parameter to 0, the Image Compression Manager determines the appropriate value for the source image. Values of 1, 2, 4, 8, 16, 24, and 32 indicate the number of bits per pixel for color images. Values of 34, 36, and 40 indicate 2-bit, 4-bit, and 8-bit grayscale, respectively, for grayscale images. Your program can determine which depths are supported by a given compressor by examining the compressor information structure returned by the `GetCodecInfo` (page 652) function

cType

You must set this parameter to a valid compressor type constant; see `Codec Identifiers`.

codec

Specify a particular compressor by setting this parameter to its compressor identifier. Alternatively, you may use a special identifier (see below). You can also specify a component instance. This may be useful if you have previously set some parameter on a specific instance of a codec field and want to make sure that the specified instance is used for that operation. See these constants:

spatialQuality

A pointer to a field containing a constant (see below) that defines the desired compressed image quality. The Image Compression Manager sets this field to the closest actual quality that the compressor can achieve. If you are not interested in this information, pass `NIL` in this parameter. See these constants:

`codecMinQuality`

`codecLowQuality`

`codecNormalQuality`

`codecHighQuality`

`codecMaxQuality`

`codecLosslessQuality`

temporalQuality

A pointer to a field containing a constant (see below) that defines the desired temporal quality. Use this value only with images that are part of image sequences. The Image Compression Manager sets this field to the closest actual quality that the compressor can achieve. If you are not interested in this information, pass `NIL` in this parameter.

compressTime

A pointer to a field to receive the compression time in milliseconds. If the compressor cannot determine the amount of time required to compress the image or if the compressor does not support this function, this field is set to 0. If you are not interested in this information, pass `NIL` in this parameter.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function allows you to verify that the quality settings you desire are supported by a given compressor component. You specify the compression characteristics, including compression type and quality, along with the image. The Image Compression Manager returns the maximum compression time for the specified image and parameters. Note that some compressors may not support this function. If the component you specify does not support this function, the Image Compression Manager returns a time value of 0.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CompressMovies

DigitizerShell

DragAndDrop Shell

MovieGWorlds

QT Internals

Declared In

`ImageCompression.h`

GetCSequenceDataRateParams

Obtains the data rate parameters previously set with SetCSequenceDataRateParams.

```
OSErr GetCSequenceDataRateParams (
   ImageSequence seqID,
   DataRateParamsPtr params
);
```

Parameters

`seqID`

Contains the unique sequence identifier that was returned by `CompressSequenceBegin` (page 609).

`params`

Points to the `data` rate parameters structure associated with the sequence identifier specified in the `seqID` parameter.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GetCSequenceFrameNumber

Returns the current frame number of the specified sequence.

```
OSErr GetCSequenceFrameNumber (
   ImageSequence seqID,
   long *frameNumber
);
```

Parameters

`seqID`

Contains the unique sequence identifier that was returned by the `CompressSequenceBegin` (page 609) function.

frameNumber
A pointer to the current frame number of the sequence identified by the `seqID` parameter.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GetCSequenceKeyFrameRate

Determines the current key frame rate of a sequence.

```
OSErr GetCSequenceKeyFrameRate (
   ImageSequence seqID,
   long *keyFrameRate
);
```

Parameters

seqID
Contains the unique sequence identifier that was returned by `CompressSequenceBegin` (page 609).

keyFrameRate
A pointer to a long integer that specifies the maximum number of frames allowed between key frames. Key frames provide points from which a temporally compressed sequence may be decompressed.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GetCSequenceMaxCompressionSize

Determines the maximum size an image will be after compression for a given compression sequence.

```
OSErr GetCSequenceMaxCompressionSize (
   ImageSequence seqID,
   PixMapHandle src,
   long *size
);
```

Parameters

seqID

Contains the unique sequence identifier that was returned by the `CompressSequenceBegin` (page 609) function.

src

A handle to the source `PixMap` structure. The compressor uses only the image's size and pixel depth to determine the maximum size of the compressed image.

size

A pointer to a field to receive the maximum size, in bytes, of the compressed image.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function is similar to `GetMaxCompressionSize` (page 670), but operates on a compression sequence instead of requiring the application to pass individual parameters about the source image.

Special Considerations

Before calling `GetCSequenceMaxCompressionSize` you must have already started a compression sequence with `CompressSequenceBegin` (page 609)

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Desktop Sprites

qteffects.win

qtsprites.win

qtwiredactions

qtwiredsprites

Declared In

`ImageCompression.h`

GetCSequencePrevBuffer

Determines the location of the previous image buffer allocated by the compressor.

```
OSErr GetCSequencePrevBuffer (
   ImageSequence seqID,
   GWorldPtr *gworld
);
```

Parameters

seqID
: Contains the unique sequence identifier that was returned by the `CompressSequenceBegin` (page 609) function.

gworld
: A pointer to a field to receive a pointer to the `CGrafPort` structure that describes the graphics world for the image buffer. You should not dispose of this graphics world; the returned pointer refers to a buffer that the Image Compression Manager is using. If the compressor has not allocated a buffer, `GetCSequencePrevBuffer` returns an error result code.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Note that this function only returns information about buffers that were allocated by the compressor. You cannot use this function to determine the location of a buffer you have provided.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GetDSequenceImageBuffer

Determines the location of the offscreen image buffer allocated by a decompressor.

```
OSErr GetDSequenceImageBuffer (
   ImageSequence seqID,
   GWorldPtr *gworld
);
```

Parameters

seqID
: Contains the unique sequence identifier that was returned by the `DecompressSequenceBegin` (page 621) function.

gworld
: A pointer to a field to receive a pointer to the `CGrafPort` structure describing the graphics world for the image buffer. You should not dispose of this graphics world; the returned pointer refers to a buffer that the Image Compression Manager is using. It is disposed of for you when the `CDSequenceEnd` (page 591) function is called. If the decompressor has not allocated a buffer, `GetDSequenceImageBuffer` returns an error result code.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GetDSequenceMatrix

Gets the matrix that was specified for a decompression sequence by a call to SetDSequenceMatrix, or that was set at DecompressSequenceBegin.

```
OSErr GetDSequenceMatrix (
   ImageSequence seqID,
   MatrixRecordPtr matrix
);
```

Parameters

seqID
: Contains the unique sequence identifier that was returned by `DecompressSequenceBegin` (page 621).

matrix
: Points to a matrix structure that specifies how to transform the image during decompression

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.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` (page 621) 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.

Availability

Available in Mac OS X v10.3 and later.

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` (page 621) 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.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`ImageCompression.h`

GetDSequenceScreenBuffer

Determines the location of the offscreen screen buffer allocated by a decompressor.

```
OSErr GetDSequenceScreenBuffer (
   ImageSequence seqID,
   GWorldPtr *gworld
);
```

Parameters

seqID

The unique sequence identifier that was returned by the DecompressSequenceBegin (page 621) function.

gworld

A pointer to a field to receive a pointer to the CGrafPort structure that describes the graphics world for the screen buffer. You should not dispose of this graphics world; the returned pointer refers to a buffer that the Image Compression Manager is using. It is disposed of for you when the CDSequenceEnd (page 591) function is called. If the decompressor has not allocated a buffer, GetDSequenceScreenBuffer returns an error result code.

Return Value

See Error Codes. Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

ImageCompression.h

GetGraphicsImporterForDataRef

Locates and opens a graphics importer component that can be used to draw the image from specified data reference.

```
OSErr GetGraphicsImporterForDataRef (
   Handle dataRef,
   OSType dataRefType,
   ComponentInstance *gi
);
```

Parameters

dataRef

The data reference to be drawn using a graphics importer component.

dataRefType

The type of data reference pointed to by the dataRef parameter; see Data References. For alias-based data references, the dataRef handle contains an AliasRecord and dataRefType is set to rAliasType.

gi

On return, contains a pointer to the ComponentInstance of the graphics importer. If no graphics importer can be found, this parameter will be set to NIL. If GetGraphicsImporterForDataRef is able to locate a graphics importer for the data reference, the returned graphics importer ComponentInstance will already be set up to draw from the specified data reference to the current port.

Return Value

See Error Codes. Returns noErr if there is no error.

Discussion

This function tries to locate a graphics importer component for the specified data reference by checking the file extension (such as .GIF or .JPG), the Macintosh file type, and the MIME type of the file. The file extension is retrieved from the data reference by using DataHGetFileName (page 783) to call the data handler associated with the data reference. If a graphics importer cannot be found using the file's type, file extension, or MIME type, GetGraphicsImporterForDataRef asks each graphics importer to validate the file, until it either finds an importer that can handle the file or exhausts the list of possible importers. This validation attempt can be quite time-consuming; to bypass it, call GetGraphicsImporterForDataRefWithFlags (page 664) instead.

Special Considerations

The caller of GetGraphicsImporterForDataRef is responsible for closing the returned ComponentInstance using CloseComponent. You must call CloseComponent when you are finished with the importer.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

ComboBoxPrefs

Fiendishthngs

ImagesToQTMovie

TextNameTool

ThreadsImporter

Declared In

ImageCompression.h

GetGraphicsImporterForDataRefWithFlags

Locates and opens a graphics importer component for a data reference with flags that control the search process.

```
OSErr GetGraphicsImporterForDataRefWithFlags (
   Handle dataRef,
   OSType dataRefType,
   ComponentInstance *gi,
   long flags
);
```

Parameters

dataRef

The data reference to be drawn using a graphics importer component.

dataRefType

The type of data reference pointed to by the dataRef parameter; see Data References. For alias-based data references, the dataRef handle contains an AliasRecord and dataRefType is set to rAliasType.

gi

On return, contains a pointer to the `ComponentInstance` of the graphics importer. If no graphics importer can be found, this parameter will be set to `NIL`. If `GetGraphicsImporterForDataRefWithFlags` is able to locate a graphics importer for the data reference, the returned graphics importer `ComponentInstance` will already be set up to draw from the specified data reference to the current port.

flags

Contains flags (see below) that control the graphics importer search process. See these constants:
`kDontUseValidateToFindGraphicsImporter`

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function tries to locate a graphics importer component for the specified data reference by checking the file extension (such as .GIF or .JPG), the Macintosh file type, and the MIME type of the file. The file extension is retrieved from the data reference by using `DataHGetFileName` (page 783) to call the data handler associated with the data reference. If a graphics importer cannot be found using the file's type, file extension, or MIME type, this function asks each graphics importer to validate the file, until it either finds an importer that can handle the file or exhausts the list of possible importers. This validation attempt can be quite time-consuming; to bypass it, pass `kDontUseValidateToFindGraphicsImporter` in the `flags` parameter.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GetGraphicsImporterForFile

Locates and opens a graphics importer component that can be used to draw a specified file.

```
OSErr GetGraphicsImporterForFile (
   const FSSpec *theFile,
   ComponentInstance *gi
);
```

Parameters

theFile

The file to be drawn using a graphics importer component.

gi

On return, contains a pointer to the `ComponentInstance` of the graphics importer. If no graphics importer can be found for the specified file, the `gi` will be set to `NIL`. If `GetGraphicsImporterForFile` is able to locate a graphics importer for the file, the returned graphics importer `ComponentInstance` will already be set up to draw the specified file to the current port.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function first tries to locate a graphics importer component for the specified file based on its file type. If it is unable to locate a graphics importer component based on the Macintosh file type, or the call is made on a non-Macintosh file, `GetGraphicsImporterForFile` will try to locate a graphics importer component based on the file extension (such as `.JPG` or `.GIF`). If a graphics importer cannot be found using the file's type or extension, `GetGraphicsImporterForFile` asks each graphics importer to validate the file, until it either finds an importer that can handle the file or exhausts the list of possible importers. This validation attempt can be quite time-consuming. To bypass the validation attempt, call `GetGraphicsImporterForFileWithFlags` (page 666) instead. The following code sample illustrates the use of `GetGraphicsImporterForFile`:

```
// Get a graphics importer for the image file, determine the natural size
// of the image, and draw the image
// See "Discovering QuickTime," page 274
void drawFile(const FSSpec *fss, const Rect *boundsRect)
    {
        GraphicsImportComponent gi;
        GetGraphicsImporterForFile(fss, &gi);
        GraphicsImportSetBoundsRect(gi, boundsRect);
        GraphicsImportDraw(gi);
        CloseComponent(gi);
    }
```

Special Considerations

The caller of `GetGraphicsImporterForFile` is responsible for closing the returned `ComponentInstance` using `CloseComponent`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Graphic Import-Export

ImproveYourImage

qtgraphics.win

qtstreamsplicer.win

vrmakepano

Declared In

`ImageCompression.h`

GetGraphicsImporterForFileWithFlags

Locates and opens a graphics importer component for a file with flags that control the search process.

```
OSErr GetGraphicsImporterForFileWithFlags (
   const FSSpec *theFile,
   ComponentInstance *gi,
   long flags
);
```

Parameters

theFile
The file to be drawn using a graphics importer component.

gi
On return, contains a pointer to the `ComponentInstance` of the graphics importer. If no graphics importer can be found for the specified file, the `gi` will be set to `NIL`. If `GetGraphicsImporterForFileWithFlags` is able to locate a graphics importer for the file, the returned graphics importer `ComponentInstance` will already be set up to draw the specified file to the current port.

flags
Contains flags (see below) that control the graphics importer search process. See these constants:
`kDontUseValidateToFindGraphicsImporter`

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function first tries to locate a graphics importer component for the specified file based on its file type. If it is unable to locate a graphics importer component based on the Macintosh file type, or the call is made on a non-Macintosh file, `GetGraphicsImporterForFile` will try to locate a graphics importer component based on the file extension (such as .JPG or .GIF). If a graphics importer cannot be found using the file's type or extension, `GetGraphicsImporterForFile` asks each graphics importer to validate the file, until it either finds an importer that can handle the file or exhausts the list of possible importers. This validation attempt can be quite time-consuming. To bypass the validation attempt, pass `kDontUseValidateToFindGraphicsImporter` in the `flags` parameter.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GetImageDescriptionCTable

Gets the custom color table for an image.

```
OSErr GetImageDescriptionCTable (
   ImageDescriptionHandle desc,
   CTabHandle *ctable
);
```

Parameters

desc
A handle to the appropriate `ImageDescription` structure.

ctable

A pointer to a field that is to receive a color table handle.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function returns the color table for the image described by the `ImageDescription` structure that is referred to by the `desc` parameter. The function correctly sizes the handle for the color table it returns.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Graphic Import-Export

ImproveYourImage

Declared In

`ImageCompression.h`

GetImageDescriptionExtension

Returns a new handle with the data from a specified image description extension.

```
OSErr GetImageDescriptionExtension (
   ImageDescriptionHandle desc,
   Handle *extension,
   long idType,
   long index
);
```

Parameters

desc

A handle to the appropriate `ImageDescription` structure.

extension

A pointer to a field to receive a handle to the returned data. The `GetImageDescriptionExtension` function returns the extended data for the image described by the `ImageDescription` structure referred to by the `desc` parameter. The function correctly sizes the handle for the data it returns.

idType

Specifies the extension's type value. Use this parameter to determine the `data` type of the `extension`. This parameter contains a four-character code, similar to an `OSType` field value.

index

The index of the extension to retrieve. This is a number between 1 and the count returned by `CountImageDescriptionExtensionType` (page 618).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function allows the application to get a copy of a specified image description extension. Note that each compressor type may have its own format for the extended data that is stored with an image. The extended data is similar in concept to the user data that applications can associate with QuickTime movies. Once you have added extended data to an image, you cannot delete it.

Special Considerations

The Image Compression Manager allocates a new handle and passes it back in the `extension` parameter. Your application should dispose of the handle when it is no longer needed.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

ImproveYourImage

Declared In

`ImageCompression.h`

GetMatrixType

Obtains information about a matrix.

```
short GetMatrixType (
   const MatrixRecord *m
);
```

Parameters

m

Points to the `MatrixRecord` structure for this operation.

Return Value

A constant (see below) that defines the type of the matrix.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

AlwaysPreview

Graphic Import-Export

ImproveYourImage

Declared In

`ImageCompression.h`

GetMaxCompressionSize

Determines the maximum size an image will be after compression.

```
ComponentResult ADD_IMAGECODEC_BASENAME() GetMaxCompressionSize
```

Parameters

src

A handle to the source image. The source image must be stored in a pixel map structure. The compressor uses only the image's size and pixel depth to determine the maximum size of the compressed image.

srcRect

A pointer to a rectangle defining the portion of the source image that is to be compressed. You may set this parameter to `NIL` if you are interested only in information about quality settings. `GetCompressionTime` (page 655) then uses the bounds of the source pixel map.

colorDepth

The depth at which the image is to be compressed. If you set this parameter to 0, the Image Compression Manager determines the appropriate value for the source image. Values of 1, 2, 4, 8, 16, 24, and 32 indicate the number of bits per pixel for color images. Values of 34, 36, and 40 indicate 2-bit, 4-bit, and 8-bit grayscale, respectively, for grayscale images. Your program can determine which depths are supported by a given compressor by examining the compressor information structure returned by `GetCodecInfo` (page 652).

quality

A constant (see below) that defines the desired compressed image quality. See these constants:

`codecMinQuality`

`codecLowQuality`

`codecNormalQuality`

`codecHighQuality`

`codecMaxQuality`

`codecLosslessQuality`

cType

You must set this parameter to a valid compressor type constant; see `Codec Identifiers`.

codec

A compressor identifier. Specify a particular compressor by setting this parameter to its compressor identifier. Alternatively, you may use a special identifier (see below). You can also specify a component instance. This may be useful if you have previously set some parameter on a specific instance of a codec field and want to make sure that the specified instance is used for that operation. See these constants:

size

A pointer to a field to receive the size, in bytes, of the compressed image.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function returns the maximum resulting size for the specified image and parameters. Your application may then use this information to allocate memory for the compression operation. The following code sample illustrates its use:

```
// GetMaxCompressionSize coding example
// See "Discovering QuickTime," page 286
PicHandle GetQTCompressedPict (PixMapHandle hpmImage)
{
    long                    lMaxCompressedSize =0;
    Handle                  hCompressedData =NIL;
    Ptr                     pCompressedData;
    ImageDescriptionHandle  hImageDesc =NIL;
    OSErr                   nErr;
    PicHandle               hpicPicture =NIL;
    Rect                    rectImage =(**hpmImage).bounds;
    CodecType               dwCodecType =kJPEGCodecType;
    CodecComponent          codec =(CodecComponent)anyCodec;
    CodecQ                  dwSpatialQuality =codecNormalQuality;
    short                   nDepth =0;          // let ICM choose depth
    nErr =GetMaxCompressionSize(hpmImage, &rectImage, nDepth,
                                        dwSpatialQuality,
                                        dwCodecType,
                                        (CompressorComponent)codec,
                                        &lMaxCompressedSize);
    if (nErr !=noErr)
        return NIL;

    hImageDesc =(ImageDescriptionHandle)NewHandle(4);
    hCompressedData =NewHandle(lMaxCompressedSize);
    if ((hCompressedData !=NIL) && (hImageDesc !=NIL)) {
        MoveHHi(hCompressedData);
        HLock(hCompressedData);
        pCompressedData =StripAddress(*hCompressedData);

        nErr =CompressImage(hpmImage,
                                        &rectImage,
                                        dwSpatialQuality,
                                        dwCodecType,
                                        hImageDesc,
                                        pCompressedData);

        if (nErr ==noErr) {
            ClipRect(&rectImage);
            hpicPicture =OpenPicture(&rectImage);
            nErr =DecompressImage(pCompressedData,
                                        hImageDesc,
                                        hpmImage,
                                        &rectImage,
                                        &rectImage,
                                        srcCopy,
                                        NIL);
            ClosePicture();
        }
        if (theErr || (GetHandleSize((Handle)hpicPicture) ==
                                            sizeof(Picture))) {
            KillPicture(hpicPicture);
            hpicPicture =NIL;
        }
    }
    if (hImageDesc !=NIL)
        DisposeHandle((Handle)hImageDesc);
    if (hCompressedData !=NIL)
        DisposeHandle(hCompressedData);
    return hpicPicture;
}
```

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
BurntTextSampleCode

Fiendishthngs

Inside Mac ICM Code

vrmakepano

VRMakePano Library

Declared In
`ImageCompression.h`

GetNextImageDescriptionExtensionType

Retrieves an image description structure extension type.

```
OSErr GetNextImageDescriptionExtensionType (
   ImageDescriptionHandle desc,
   long *idType
);
```

Parameters

desc

A handle to an `ImageDescription` structure.

idType

A pointer to an integer that indicates the type of the extension after which this function is to return the `next` extension type. `Point` to a value of 0 to return the first type found.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function allows your application to search for all the types of extensions in an `ImageDescription` structure. The `idType` parameter should be set to 0 to start the search. When no more extension types can be found, the function will set this field to 0.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GetPictureFileHeader

Extracts the picture frame and file header from a specified picture file.

```
OSErr GetPictureFileHeader (
   short refNum,
   Rect *frame,
   OpenCPicParams *header
);
```

Parameters

refNum

A file reference number for the source PICT file.

frame

A pointer to a rectangle that is to receive the picture frame rectangle of the picture file. This function places the `picFrame` rectangle from the picture structure into the rectangle referred to by the `frame` parameter. If you are not interested in this information, pass `NIL` in this parameter.

header

A pointer to an `OpenCPicParams` structure. The `GetPictureFileHeader` function places the header from the specified picture file into this structure. If you are not interested in this information, pass `NIL` in this parameter.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Your program can use the information returned in the `header` parameter to determine how to draw an image without having to read the picture file.

Special Considerations

Note that this function always returns a version 2 header. If the source file is a version 1 PICT file, the `GetPictureFileHeader` function converts the header into version 2 format before returning it to your application. See *Inside Macintosh: Imaging With QuickDraw* for more information about picture headers.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

DrawTextCodec

Declared In

`ImageCompression.h`

GetSimilarity

Compares a compressed image to a picture stored in a pixel map and returns a value indicating the relative similarity of the two images.

```
ComponentResult ADD_IMAGECODEC_BASENAME() GetSimilarity
```

Parameters

src

A handle to the noncompressed image. The image must be stored in a pixel map structure.

srcRect

A pointer to a rectangle defining the portion of the image to compare to the compressed image. This rectangle should be the same size as the image described by the `ImageDescription` structure specified by the `desc` parameter.

desc

A handle to the `ImageDescription` structure that defines the compressed image for the operation.

data

Points to the compressed image data. This pointer must contain a 32-bit clean address.

similarity

A pointer to a field that is to receive the similarity value. The compressor sets this field to reflect the relative similarity of the two images. Valid values range from 0 (completely different) to 1.0 (identical).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

HitTestDSequenceData

Undocumented

```
OSErr HitTestDSequenceData (
   ImageSequence seqID,
   void *data,
   Size dataSize,
   Point where,
   long *hit,
   long hitFlags
);
```

Parameters

seqID

The unique sequence identifier that was returned by the `DecompressSequenceBegin` (page 621) function.

data

A pointer to data.

dataSize

The size of the data.

where

A `Point` structure that defines the hit location.

hit

Undocumented

hitFlags

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

ICMDecompressComplete

Signals the completion of a decompression operation.

```
void ICMDecompressComplete (
   ImageSequence seqID,
   OSErr err,
   short flag,
   ICMCompletionProcRecordPtr completionRtn
);
```

Parameters

seqID

The unique sequence identifier assigned by `CompressSequenceBegin` (page 609) or `DecompressSequenceBegin` (page 621).

err

Indicates whether the operation succeeded or failed. Set this parameter to 0 for successful operations. For failed operations, set the error code appropriate for the failure. For canceled operations (for example, when the ICM calls your component's `ImageCodecFlush` (page 891) function), set this parameter to -1.

flag

Completion flags (see below). Note that you may set more than one of these flags to 1. See these constants:

`codecCompletionSource`

`codecCompletionDest`

`codecCompletionDontUnshield`

completionRtn

A pointer to an `ICMCompletionProcRecord` structure. That structure identifies the application's completion function and contains a reference constant associated with the frame. Your component obtains this structure as part of the `CodecDecompressParams` structure provided by the Image Compression Manager at the start of the decompression operation.

Discussion

Your component must call this function at the end of decompression operations.

Special Considerations

Prior to QuickTime 2.0, decompressor components called the application's completion function directly. For compatibility, that method is still supported except for scheduled asynchronous decompression operations, which must use the `ICMDecompressComplete` call. Newer decompressors should always use `ICMDecompressComplete` rather than calling the completion function directly, regardless of the type of decompression operation.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

ICMDecompressCompleteS

Undocumented

```
OSErr ICMDecompressCompleteS (
   ImageSequence seqID,
   OSErr err,
   short flag,
   ICMCompletionProcRecordPtr completionRtn
);
```

Parameters

`seqID`

The unique sequence identifier assigned by `CompressSequenceBegin` (page 609) or `DecompressSequenceBegin` (page 621).

`err`

Indicates whether the operation succeeded or failed. See `Error Codes`.

`flag`

Undocumented

`completionRtn`

A pointer to an `ICMCompletionProcRecord` structure.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

ICMGetPixelFormatInfo

Retrieves pixel format information.

```
OSErr ICMGetPixelFormatInfo (
   OSType PixelFormat,
   ICMPixelFormatInfoPtr theInfo
);
```

Parameters

PixelFormat

A constant that identifies the format; see `Pixel Formats`.

theInfo

A pointer to your `ICMPixelFormatInfo` structure in which information is returned. You should initialize the `size` field of this structure with `sizeof (ICMPixelFormatInfo)`. The function will not copy more than this number of bytes into the structure. On return, the `size` field contains the actual size of the data structure. If this amount is greater the size you passed in, that means you didn't retrieve all of the information.

Return Value

Returns `cDepthErr` if the pixel format is not valid. For other errors, see `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

ICMSequenceGetChainMember

Undocumented

```
OSErr ICMSequenceGetChainMember (
   ImageSequence seqID,
   ImageSequence *retSeqID,
   long flags
);
```

Parameters

seqID

The unique sequence identifier assigned by `CompressSequenceBegin` (page 609) or `DecompressSequenceBegin` (page 621).

retSeqID

Undocumented

flags

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

ICMSequenceGetInfo

Gets multiprocessing properties for compression and decompression sequences.

```
OSErr ICMSequenceGetInfo (
   ImageSequence seqID,
   OSType which,
   void *data
);
```

Parameters

seqID
The unique sequence identifier assigned by `CompressSequenceBegin` (page 609) or `DecompressSequenceBegin` (page 621).

which
A constant (see below) that determines the property to be returned. See these constants:
`kICMSequenceTaskWeight`
`kICMSequenceTaskName`

data
The value of the property indicated by the which parameter.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function determines if ICM clients have requested that multiprocessor tasks assisting compression and decompression operations use specific task weights and task names.

Special Considerations

Apple's multiprocessing capability supports both co-operatively scheduled tasks and preemptively scheduled tasks. The support for preemptively tasks allow applications to create symmetrically scheduled preemptive tasks that can be run on a single processor machine, and will take full advantage of multiple processors when they are installed.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

ICMSequenceLockBits

Undocumented

```
OSErr ICMSequenceLockBits (
   ImageSequence seqID,
   PixMapPtr dst,
   long flags
);
```

Parameters

seqID

The unique sequence identifier assigned by CompressSequenceBegin (page 609) or DecompressSequenceBegin (page 621).

dst

A pointer to a PixMap structure.

flags

Undocumented

Return Value

See Error Codes. Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

ImageCompression.h

ICMSequenceSetInfo

Sets multiprocessing properties for compression and decompression sequences.

```
OSErr ICMSequenceSetInfo (
   ImageSequence seqID,
   OSType which,
   void *data,
   Size dataSize
);
```

Parameters

seqID

The unique sequence identifier assigned by CompressSequenceBegin (page 609) or DecompressSequenceBegin (page 621).

which

A constant (see below) that determines the property to be set. See these constants:

kICMSequenceTaskWeight

kICMSequenceTaskName

data

The value of the property to be set.

dataSize

The length in bytes of the `data` parameter.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function lets ICM clients request that multiprocessor tasks assisting compression and decompression operations use specific task weights and task names.

Special Considerations

Apple's multiprocessing capability supports both co-operatively scheduled tasks and preemptively scheduled tasks. The support for preemptively tasks allow applications to create symmetrically scheduled preemptive tasks that can be run on a single processor machine, and will take full advantage of multiple processors when they are installed.

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

ICMSequenceUnlockBits

Undocumented

```
OSErr ICMSequenceUnlockBits (
   ImageSequence seqID,
   long flags
);
```

Parameters

seqID

The unique sequence identifier assigned by `CompressSequenceBegin` (page 609) or `DecompressSequenceBegin` (page 621).

flags

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

ICMSetPixelFormatInfo

Lets you define your own pixel format.

```
OSErr ICMSetPixelFormatInfo (
   OSType PixelFormat,
   ICMPixelFormatInfoPtr theInfo
);
```

Parameters

PixelFormat

A pixel format constant. See `Pixel Formats`.

theInfo

A pointer to an `ICMPixelFormatInfo` structure containing a definition of the new pixel format.

Return Value

Returns `paramErr` if the format is already defined. See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

OpenGLCompositorLab

SoftVideoOutputComponent

Declared In

`ImageCompression.h`

ICMShieldSequenceCursor

Hides the cursor during decompression operations.

```
OSErr ICMShieldSequenceCursor (
   ImageSequence seqID
);
```

Parameters

seqID

The unique sequence identifier, assigned by `CompressSequenceBegin` (page 609) or `DecompressSequenceBegin` (page 621), for which to shield the cursor.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

For correct image display behavior, the cursor must be shielded (hidden) during decompression. By default, the Image Compression Manager handles the cursor for you, hiding it at the beginning of a decompression operation and revealing it at the end. With scheduled asynchronous decompression, however, the ICM cannot do as precise a job of managing the cursor, because it does not know exactly when scheduled operations actually begin and end. While the ICM can still manage the cursor, it must hide the cursor when each request is queued, rather than when the request is serviced. This may result in the cursor remaining hidden for long periods of time. To achieve better cursor behavior, you can choose to manage the cursor in your decompressor component. If you so choose, you can use this function to hide the cursor; the ICM displays the cursor when you call `ICMDecompressComplete` (page 675). In this manner, the cursor is hidden only when your component is decompressing and displaying the frame.

Special Considerations

This function may be called at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

ImageFieldSequenceBegin

Initiates an image field sequence operation and specifies the input and output data format.

```
OSErr ImageFieldSequenceBegin (
   ImageFieldSequence *ifs,
   ImageDescriptionHandle desc1,
   ImageDescriptionHandle desc2,
   ImageDescriptionHandle descOut
);
```

Parameters

ifs

On return, contains the unique sequence identifier assigned to the sequence.

desc1

An `ImageDescription` structure describing the format and characteristics of the data to be passed to `ImageFieldSequenceExtractCombine` (page 683) through the data1 parameter.

desc2

An `ImageDescription` structure describing the format and characteristics of the data to be passed to the `ImageFieldSequenceExtractCombine` function through the data2 parameter. Set to `NIL` if the requested operation uses only one input frame.

descOut

The desired format of the resulting frames. Typically this is the same format specified by the desc1 and desc2 parameters.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Use this function to set up an image field sequence operation and specify the input and output data format.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

ImageFieldSequenceEnd

Ends an image field sequence operation.

```
OSErr ImageFieldSequenceEnd (
   ImageFieldSequence ifs
);
```

Parameters

ifs

The unique sequence identifier that was returned by the `ImageFieldSequenceBegin` (page 682) function.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

You must call this function to terminate an image field sequence operation.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

ImageFieldSequenceExtractCombine

Performs field operations on video data.

```
OSErr ImageFieldSequenceExtractCombine (
   ImageFieldSequence ifs,
   long fieldFlags,
   void *data1,
   long dataSize1,
   void *data2,
   long dataSize2,
   void *outputData,
   long *outDataSize
);
```

Parameters

ifs

The unique sequence identifier that was returned by `ImageFieldSequenceBegin` (page 682).

fieldFlags
Flags (see below) that specify the operation to be performed. A correctly formed request will specify two input fields, mapping one to the odd output field and the other to the even output field. See these constants:

```
evenField1ToEvenFieldOut
evenField1ToOddFieldOut
oddField1ToEvenFieldOut
oddField1ToOddFieldOut
evenField2ToEvenFieldOut
evenField2ToOddFieldOut
oddField2ToEvenFieldOut
oddField2ToOddFieldOut
```

data1
A pointer to a buffer containing the `data` of input field one.

dataSize1
The size of the data1 buffer.

data2
A pointer to a buffer containing the `data` of input field two. Set to `NIL` if the requested operation uses only one input frame.

dataSize2
The size of the data2 buffer. Set to 0 if the requested operation uses only one input frame.

outputData
A pointer to a buffer to receive the resulting frame. Use `GetMaxCompressionSize` (page 670) to determine the amount of memory to allocate for this buffer.

outDataSize
On output this parameter returns the actual size of the data.

Return Value
Returns the `codecUnimpErr` result code if there is no codec present in the system that can perform the requested operation. See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function provides a method for working directly with fields of interlaced video. You can use it to change the field dominance of an image by reversing the two fields, or to create or remove the effects of the 3:2 pulldown commonly performed when transferring film to NTSC videotape. Because this function operates directly on the compressed video data, it is faster than working with decompressed images. It also has the added benefit of eliminating any image quality degradation that might result from lossy codecs.

This function accepts one or two compressed images as input and creates a single compressed image on output. You specify the operation to be performed using the `fieldFlags` parameter.

Special Considerations

The Apple Component Video (YUV) and Motion JPEG codecs currently support this function.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

ImageTranscodeDisposeFrameData

Disposes transcoded image data.

```
OSErr ImageTranscodeDisposeFrameData (
   ImageTranscodeSequence its,
   void *dstData
);
```

Parameters

`its`
: The image transcoder sequence that was used to generate the transcoded data.

`dstData`
: A pointer to the transcoded image data generated by the `ImageTranscodeFrame` (page 685) function.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
When the transcoded image data returned by `ImageTranscodeFrame` (page 685) is no longer needed, use this function to dispose of the data. Only the image transcoder that generated the data can properly dispose of it.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

ImageTranscodeFrame

Transcodes a frame of image data.

```
OSErr ImageTranscodeFrame (
   ImageTranscodeSequence its,
   void *srcData,
   long srcDataSize,
   void **dstData,
   long *dstDataSize
);
```

Parameters

`its`
: The image transcoder sequence to use to perform the transcoding operation.

`srcData`
: A pointer to the source data to transcode.

srcDataSize

The size of the compressed source image data in bytes.

dstData

On return, a pointer to the transcoded image data.

dstDataSize

On return, the size of the transcoded image data.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

After creating the image transcoder sequence, using `ImageTranscodeSequenceBegin` (page 686), use this function to transcode a frame of image data. The caller is responsible for disposing of the transcoded data using `ImageTranscodeDisposeFrameData` (page 685).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

ImageTranscodeSequenceBegin

Initiates an image transcoder sequence operation.

```
OSErr ImageTranscodeSequenceBegin (
   ImageTranscodeSequence *its,
   ImageDescriptionHandle srcDesc,
   OSType destType,
   ImageDescriptionHandle *dstDesc,
   void *data,
   long dataSize
);
```

Parameters

its

The image transcoder sequence identifier. If the operation fails, the value pointed to is set to `NIL`.

srcDesc

The `ImageDescription` structure for the source compressed image data.

destType

The desired compression format into which to transcode the source data.

dstDesc

On return, an `ImageDescription` structure for the data which will be generated by the image transcoding sequence.

data

A pointer to first frame of compressed data to transcode. Set to `NIL` of not available.

dataSize

The size of the compressed data, in bytes. Set to 0 if no data is provided.

Return Value
See `Error Codes`. Returns `noErr` if there is no error. If no transcoder is available to perform the requested transcoding operation, a `cantFindHandler` error is returned.

Discussion
This function begins an image transcoder sequence operation and returns the sequence identifier in the `its` parameter. The caller is responsible for disposing of the `ImageDescription` structure that is returned in the `dstDesc` parameter.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

ImageTranscodeSequenceEnd

Ends an image transcoder sequence operation.

```
OSErr ImageTranscodeSequenceEnd (
   ImageTranscodeSequence its
);
```

Parameters

`its`
The identifier of the image transcoder sequence to dispose. It is safe to pass a value of 0 in this parameter.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
You must call this function to terminate an image transcoder sequence operation and dispose of the sequence.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

InverseMatrix

Creates a new matrix that is the inverse of a specified matrix.

```
Boolean InverseMatrix (
   const MatrixRecord *m,
   MatrixRecord *im
);
```

Parameters

m

A pointer to the source `MatrixRecord` structure for the operation.

im

A pointer to a `MatrixRecord` structure that is to receive the new matrix. The function updates this structure so that it contains a matrix that is the inverse of that specified by the `m` parameter.

Return Value

A Boolean value of TRUE if `InverseMatrix` was able to create an inverse matrix, FALSE otherwise.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qtactiontargets

qtactiontargets.win

qtwiredspritesjr

qtwiredspritesjr.win

Declared In

`ImageCompression.h`

MakeFilePreview

Creates a preview for a file.

```
OSErr MakeFilePreview (
   short resRefNum,
   ICMProgressProcRecordPtr progress
);
```

Parameters

resRefNum

The resource file for this operation. You must have opened this resource file with write permission. If there is a preview in the specified file, the Movie Toolbox replaces that preview with a new one.

progress

A pointer to an `ICMProgressProcRecord` structure. During the process of creating the preview, the Movie Toolbox may occasionally call a function you provide in order to report its progress. You can then use this information to keep the user informed.

Set this parameter to -1 to use the default progress function. If you specify a progress function, it must comply with the interface defined for Image Compression Manager progress functions; see "Image Compression Manager" in *Inside Macintosh: QuickTime* for more information. Set this parameter to `NIL` to prevent the Movie Toolbox from calling a progress function.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
You should create a preview whenever you save a movie. You specify the file by supplying a reference to its resource file. You must have opened this resource file with write permission. If there is a preview in the specified file, the Movie Toolbox replaces that preview with a new one.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qtinfo

qtinfo.win

Declared In
`ImageCompression.h`

MakeImageDescriptionForEffect

Returns an ImageDescription structure you can use to help create a sample description for an effect.

```
OSErr MakeImageDescriptionForEffect (
   OSType effectType,
   ImageDescriptionHandle *idh
);
```

Parameters

`effectType`
The four-character code identifying the type of effect to make an image description for. See `Effects Codes`.

`idh`
The handle of an `ImageDescription` structure. On entry, this parameter normally points to an `ImageDescription` structure whose contents are `NIL`. On return, the structure is correctly filled out for the selected effect type.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
To create a sample description, you create and fill out a data structure of type `ImageDescription`. This function simplifies this process. Only sample descriptions made with this function can be used in stacked effects, where one effect track acts as a source for another.

The following sample code creates a sample description:.

```
// MakeImageDescriptionForEffect coding example
// Return a new image description with default and specified values.
ImageDescriptionHandle QTEffSeg_MakeSampleDescription (
                   OSType theEffectType, short theWidth, short theHeight)
{
    ImageDescriptionHandle      mySampleDesc =NIL;
#if USES_MAKE_IMAGE_DESC_FOR_EFFECT
    OSErr  myErr =noErr;
    // create a new sample description
    myErr =MakeImageDescriptionForEffect(theEffectType, &mySampleDesc);
    if (myErr !=noErr)
        return(NIL);
#else
    // create a new sample description
    mySampleDesc =(ImageDescriptionHandle)
                                 NewHandleClear(sizeof(ImageDescription));
    if (mySampleDesc ==NIL)
        return(NIL);

    // fill in the fields of the sample description
    (**mySampleDesc).cType =theEffectType;
    (**mySampleDesc).idSize =sizeof(ImageDescription);
    (**mySampleDesc).hRes =72L << 16;
    (**mySampleDesc).vRes =72L << 16;
    (**mySampleDesc).frameCount =1;
    (**mySampleDesc).depth =0;
    (**mySampleDesc).clutID =-1;
#endif

    (**mySampleDesc).vendor =kAppleManufacturer;
    (**mySampleDesc).temporalQuality =codecNormalQuality;
    (**mySampleDesc).spatialQuality =codecNormalQuality;
    (**mySampleDesc).width =theWidth;
    (**mySampleDesc).height =theHeight;

    return(mySampleDesc);
}
```

Version Notes
Introduced in QuickTime 4. Image descriptions built using sample code from earlier versions of QuickTime cannot be used when stacking effects.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
makeeffectslideshow

qtaddeffectseg.win

qteffects.win

qtshoweffect

samplemakeeffectmovie.win

Declared In
`ImageCompression.h`

MakeImageDescriptionForPixMap

Fills out an ImageDescription structure corresponding to a PixMap structure.

```
OSErr MakeImageDescriptionForPixMap (
   PixMapHandle pixmap,
   ImageDescriptionHandle *idh
);
```

Parameters

pixmap

A handle to a `PixMap` structure.

idh

The handle of an `ImageDescription` structure. On entry, this parameter normally points to an `ImageDescription` structure whose contents are `NIL`. On return, the structure is correctly filled out for the selected `PixMap`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

ImproveYourImage

SGDataProcSample

VideoProcessing

vrscript

vrscript.win

Declared In

`ImageCompression.h`

MakeThumbnailFromPicture

Creates a thumbnail picture from a specified Picture structure.

```
OSErr MakeThumbnailFromPicture (
   PicHandle picture,
   short colorDepth,
   PicHandle thumbnail,
   ICMProgressProcRecordPtr progressProc
);
```

Parameters

picture

A handle to the image from which the thumbnail is to be extracted. The image must be stored in a `Picture` structure.

colorDepth

The depth at which the image is likely to be viewed. If you set this parameter to 0, the Image Compression Manager determines the appropriate value for the source image. Values of 1, 2, 4, 8, 16, 24, and 32 indicate the number of bits per pixel for color images. Values of 34, 36, and 40 indicate 2-bit, 4-bit, and 8-bit grayscale, respectively, for grayscale images.

thumbnail
  A handle to the destination `Picture` structure for the thumbnail image. The compressor resizes this handle for the resulting data.

progressProc
  A pointer to an `ICMProgressProcRecord` structure. During the operation, the Image Compression Manager will occasionally call a function to report its progress. You can provide a function through this structure. If you have not provided a progress function, set this parameter to `NIL`. If you pass a value of -1, you obtain a standard progress function.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qtinfo
qtinfo.win

Declared In
`ImageCompression.h`

MakeThumbnailFromPictureFile

Creates a thumbnail picture from a specified picture file.

```
OSErr MakeThumbnailFromPictureFile (
   short refNum,
   short colorDepth,
   PicHandle thumbnail,
   ICMProgressProcRecordPtr progressProc
);
```

Parameters

refNum
  A file reference number for the PICT file from which the thumbnail is to be extracted.

colorDepth
  The depth at which the image is likely to be viewed. If you set this parameter to 0, the Image Compression Manager determines the appropriate value for the source image. Values of 1, 2, 4, 8, 16, 24, and 32 indicate the number of bits per pixel for color images. Values of 34, 36, and 40 indicate 2-bit, 4-bit, and 8-bit grayscale, respectively, for grayscale images.

thumbnail
  A handle to the destination picture structure for the thumbnail image. The compressor resizes this handle for the resulting data.

progressProc
  A pointer to an `ICMProgressProcRecord` structure. During the operation, the Image Compression Manager will occasionally call a function to report its progress. You can provide a function through this structure.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

MakeThumbnailFromPixMap

Creates a thumbnail picture from a specified PixMap structure.

```
OSErr MakeThumbnailFromPixMap (
   PixMapHandle src,
   const Rect *srcRect,
   short colorDepth,
   PicHandle thumbnail,
   ICMProgressProcRecordPtr progressProc
);
```

Parameters

`src`

A handle to the image from which the thumbnail is to be extracted. The image must be stored in a `PixMap` structure.

`srcRect`

A pointer to a `Rect` structure that defines the portion of the image to use for the thumbnail.

`colorDepth`

The depth at which the image is likely to be viewed. If you set this parameter to 0, the Image Compression Manager determines the appropriate value for the source image. Values of 1, 2, 4, 8, 16, 24, and 32 indicate the number of bits per pixel for color images. Values of 34, 36, and 40 indicate 2-bit, 4-bit, and 8-bit grayscale, respectively, for grayscale images.

`thumbnail`

A handle to the destination picture structure for the thumbnail image. The compressor resizes this handle for the resulting data.

`progressProc`

A pointer to an `ICMProgressProcRecord` structure. During the operation, the Image Compression Manager will occasionally call a function to report its progress. You can provide a function through this structure.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

MapMatrix

Alters an existing matrix so that it defines a transformation from one rectangle to another.

```
void MapMatrix (
   MatrixRecord *matrix,
   const Rect *fromRect,
   const Rect *toRect
);
```

Parameters

matrix
A pointer to a matrix structure. The `MapMatrix` function modifies this matrix so that it performs a transformation in the rectangle specified by the `toRect` parameter that is analogous to the transformation it currently performs in the rectangle specified by the `fromRect` parameter.

fromRect
A pointer to the source `Rect` structure.

toRect
A pointer to the destination `Rect` structure.

Discussion
`MapMatrix` affects only the scaling and translation attributes of the matrix.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTCarbonShell

vrmovies

vrmovies.win

vrscript

vrscript.win

Declared In
`ImageCompression.h`

NewImageGWorld

Creates an offscreen graphics world.

```
ComponentResult ADD_IMAGECODEC_BASENAME() NewImageGWorld
```

Parameters

gworld
A pointer to a graphic world created using the width, height, depth, and color table specified in the `ImageDescription` structure pointed to in the `idh` parameter.

idh

A handle to an `ImageDescription` structure that contains information for the graphics world pointed to by the `gworld` parameter.

flags

Graphics world creation flags (see below). The `pixPurge`, `noNewDevice`, `useTempMem`, `keepLocal`, `pixelsPurgeable`, and `pixelsLocked` flags are commands to this function; the others are returned by this function. See these constants:

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

PtInDSequenceData

Tests to see if a compressed image contains data at a a given point.

```
OSErr PtInDSequenceData (
   ImageSequence seqID,
   void *data,
   Size dataSize,
   Point where,
   Boolean *hit
);
```

Parameters

seqID

The unique sequence identifier that was returned by the `DecompressSequenceBegin` (page 621) function.

data

Pointer to compressed data in the format specified by the `desc param`.

dataSize

`Size` of the compressed data referred to by the data `param`.

where

A QuickDraw `Point` structure of value (0,0), based at the top-left corner of the image.

hit

A pointer to a field to receive the Boolean indicating whether or not the image contained data at the specified point. The Boolean will be set to TRUE if the point specified by the `where` parameter is contained within the compressed image data specified by the data `param`, or FALSE if the specified point falls within a blank portion of the image.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion

`PtInDSequenceData` allows the application to perform hit testing on compressed data.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

QTGetFileNameExtension

Gets the extension to a file name.

```
OSErr QTGetFileNameExtension (
   ConstStrFileNameParam fileName,
   OSType fileType,
   OSType *extension
);
```

Parameters

fileName

A file name string.

fileType

A file type; see `File Types and Creators`.

extension

A pointer to the file name extension string.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qtdataexchange

qtdataexchange.win

Declared In

`ImageCompression.h`

QTGetPixelFormatDepthForImageDescription

For a given pixel format, returns the depth value that should be used in image descriptions.

```
short QTGetPixelFormatDepthForImageDescription (
   OSType PixelFormat
);
```

Parameters

PixelFormat

The image description's pixel format; see `Pixel Formats`.

Return Value

The pixel depth for that format.

Discussion

Given a pixel format, this function returns the corresponding depth value that should be used in image descriptions. Such a value is not the literal number of bits per pixel, but the closest corresponding classic QuickDraw depth. For any pixel format with an alpha channel, it is 32. For grayscale pixel formats of 8 or more bits per pixel, it is 40. For color quantized to 5 or 6 bits per component, it is 16. For all other color pixel formats, it is 24.

Version Notes

Introduced in QuickTime 6.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`ImageCompression.h`

QTGetPixelSize

Returns the bits per pixel for a given pixel format.

```
short QTGetPixelSize (
   OSType PixelFormat
);
```

Parameters

PixelFormat

A constant that identifies the pixel format; see `Pixel Formats`. This function returns meaningful information only for non-planar formats.

Return Value

The bits per pixel. Returns 0 if the format is unknown.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Dimmer2Effect

Dimmer2Effect.win

ElectricImageComponent

ElectricImageComponent.win

GreyscaleEffectSample

Declared In
`ImageCompression.h`

QTGetPixMapHandleGammaLevel

Retrieves the current PixMap extension's gamma level setting.

```
Fixed QTGetPixMapHandleGammaLevel (
   PixMapHandle pm
);
```

Parameters

pm

A handle to a `PixMap` structure that has a `PixMapExtension` structure.

Return Value
On return, the gamma level previously set (or the default level) for the pixel map referenced by the `pm` parameter.

Discussion
A typical use for this function is to retrieve the gamma level of a pixel map after a codec decompresses it into a `PixMap` structure.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

QTGetPixMapHandleRequestedGammaLevel

Retrieves the current PixMap extension's requested gamma level setting.

```
Fixed QTGetPixMapHandleRequestedGammaLevel (
   PixMapHandle pm
);
```

Parameters

pm

A handle to a `PixMap` structure that has a `PixMapExtension` structure.

Return Value
On return, the requested gamma level previously set (or the default level) for the pixel map referenced by the `pm` parameter.

Discussion
A typical use for this function is to retrieve the gamma level of a pixel map after a codec decompresses it into a `PixMap` structure. The requested gamma level is used to control what gamma conversion is attempted during decompression. The requested gamma level may differ from the actual gamma level depending on the compressed data and the capabilities of the codecs involved.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

QTGetPixMapHandleRowBytes

Gets the rowBytes value for a pixel map accessed by a handle.

```
long QTGetPixMapHandleRowBytes (
   PixMapHandle pm
);
```

Parameters
pm

A handle to a `PixMap` structure.

Return Value
The `rowBytes` value.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTCarbonShell

qteffects.win

qtsprites.win

qtwiredactions

qtwiredsprites

Declared In
`ImageCompression.h`

QTGetPixMapPtrGammaLevel

Retrieves the current PixMap extension's gamma level setting.

```
Fixed QTGetPixMapPtrGammaLevel (
   PixMapPtr pm
);
```

Parameters
pm

A pointer to a `PixMap` structure that has a `PixMapExtension` structure.

Return Value
On return, the gamma level previously set (or the default level) for the pixel map pointed to by the `pm` parameter.

Discussion
A typical use for this function is to retrieve the gamma level of a pixel map after a codec decompresses it into a `PixMap` structure.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

QTGetPixMapPtrRequestedGammaLevel

Retrieves the current PixMap extension's gamma level setting.

```
Fixed QTGetPixMapPtrRequestedGammaLevel (
   PixMapPtr pm
);
```

Parameters

pm

A pointer to a `PixMap` structure that has a `PixMapExtension` structure.

Return Value
On return, the requested gamma level previously set (or the default level) for the pixel map pointed to by the `pm` parameter.

Discussion
A typical use for this function is to retrieve the gamma level of a pixel map after a codec decompresses it into a `PixMap` structure. The requested gamma level is used to control what gamma conversion is attempted during decompression. The requested gamma level may differ from the actual gamma level depending on the compressed data and the capabilities of the codecs involved

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

QTGetPixMapPtrRowBytes

Gets the rowBytes value for a pixel map accessed by a pointer.

```
long QTGetPixMapPtrRowBytes (
   PixMapPtr pm
);
```

Parameters

pm

A pointer to a `PixMap` structure.

Return Value

The `rowBytes` value.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Dimmer2Effect

Dimmer2Effect.win

GreyscaleEffectSample

SoftVideoOutputComponent

VideoProcessing

Declared In

`ImageCompression.h`

QTNewGWorld

Creates an offscreen graphics world that may have a non-Macintosh pixel format.

```
OSErr QTNewGWorld (
   GWorldPtr *offscreenGWorld,
   OSType PixelFormat,
   const Rect *boundsRect,
   CTabHandle cTable,
   GDHandle aGDevice,
   GWorldFlags flags
);
```

Parameters

offscreenGWorld

On return, a pointer to the offscreen graphics world created by this routine.

PixelFormat

The new graphics world's pixel format; see `Pixel Formats`. This function won't work with planar pixel formats; use `QTNewGWorldFromPtr` (page 703) instead. See the `ICMPixelFormatInfo` structure for a discussion of planar and chunky formats.

boundsRect

A pointer to the boundary rectangle and port rectangle for the offscreen pixel map. This becomes the boundary rectangle for the `GDevice` structure, if this function creates one. If you specify 0 in the `PixelFormat` parameter, the function interprets the boundaries in global coordinates that it uses to determine which screens intersect the rectangle. It then uses the pixel format, color table, and `GDevice` structure from the screen with the greatest pixel depth from among all screens whose boundary rectangles intersect this rectangle. Typically, your application supplies this parameter with the port rectangle for the onscreen window into which your application will copy the pixel image from this offscreen world.

cTable

A handle to a `ColorTable` structure. If you pass `NIL` in this parameter, the function uses the default color table for the pixel format that you specify in the `PixelFormat` parameter. If you set the `PixelFormat` parameter to 0, the function ignores the `cTable` parameter and instead copies and uses the color table of the graphics device with the greatest pixel depth among all graphics devices whose boundary rectangles intersect the rectangle that you specify in the `boundsRect` parameter. If you use this function on a computer that supports only basic QuickDraw, you may specify only `NIL` in this parameter.

aGDevice

A handle to a `GDevice` structure that is used only when you specify the `noNewDevice` flag in the `flags` parameter, in which case the function attaches this structure to the new offscreen graphics world. If you set the `PixelFormat` parameter to 0, or if you do not set the `noNewDevice` flag, the function ignores this parameter, so you should set it to `NIL`. If you set the `PixelFormat` parameter to 0, the function uses the `GDevice` structure for the graphics device with the greatest pixel depth among all graphics devices whose boundary rectangles intersect the rectangle that you specify in the `boundsRect` parameter. You should pass `NIL` in this parameter if the computer supports only basic QuickDraw. Generally, your application should never create `GDevice` structures for offscreen graphics worlds.

flags

Constants (see below) that identify options available to your application. You can set a combination of these flags. If you don't wish to use any of them, pass 0 in this parameter. In this case the default behavior is to create an offscreen graphics world where the base address for the offscreen pixel image is unpurgeable, the graphics world uses an existing `GDevice` structure (if you pass 0 in the depth parameter) or creates a new `GDevice` structure, it uses memory in your application heap, and it allows graphics accelerators to cache the offscreen pixel image. See these constants:

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Desktop Sprites

qteffects

qteffects.win

qtwiredactions

VideoProcessing

Declared In
`ImageCompression.h`

QTNewGWorldFromPtr

Wraps a graphics world and pixel map structure around an existing block of memory containing an image.

```
OSErr QTNewGWorldFromPtr (
   GWorldPtr *gw,
   OSType pixelFormat,
   const Rect *boundsRect,
   CTabHandle cTable,
   GDHandle aGDevice,
   GWorldFlags flags,
   void *baseAddr,
   long rowBytes
);
```

Parameters

gw

On entry, a pointer that isn't going to change during the lifetime of the allocated graphics world. On return, this pointer references the offscreen graphics world created by this function.

pixelFormat

The new graphics world's pixel format; see `Pixel Formats`.

boundsRect

A pointer to the boundary rectangle and port rectangle for the offscreen pixel map. This becomes the boundary rectangle for the `GDevice` structure, if this function creates one. If you specify 0 in the `pixelFormat` parameter, the function interprets the boundaries in global coordinates that it uses to determine which screens intersect the rectangle. It then uses the pixel format, color table, and `GDevice` structure from the screen with the greatest pixel depth from among all screens whose boundary rectangles intersect this rectangle. Typically, your application supplies this parameter with the port rectangle for the onscreen window into which your application will copy the pixel image from this offscreen world.

cTable

A handle to a `ColorTable` structure. If you pass `NIL` in this parameter, the function uses the default color table for the pixel format that you specify in the `pixelFormat` parameter. If you set the `pixelFormat` parameter to 0, the function ignores the `cTable` parameter and instead copies and uses the color table of the graphics device with the greatest pixel depth among all graphics devices whose boundary rectangles intersect the rectangle that you specify in the `boundsRect` parameter. If you use this function on a computer that supports only basic QuickDraw, you may specify only `NIL` in this parameter.

aGDevice

A handle to a `GDevice` structure that is used only when you specify the `noNewDevice` flag in the `flags` parameter, in which case the function attaches this structure to the new offscreen graphics world. If you set the `pixelFormat` parameter to 0, or if you do not set the `noNewDevice` flag, the function ignores this parameter, so you should set it to `NIL`. If you set the `pixelFormat` parameter to 0, the function uses the `GDevice` structure for the graphics device with the greatest pixel depth among all graphics devices whose boundary rectangles intersect the rectangle that you specify in the `boundsRect` parameter. You should pass `NIL` in this parameter if the computer supports only basic QuickDraw. Generally, your application should never create `GDevice` structures for offscreen graphics worlds.

flags

A constant (see below) that identifies an option available to your application. If you don't wish to use this option, pass 0 in this parameter. In this case the default behavior is to create an offscreen graphics world that uses an existing `GDevice` structure (if you pass 0 in the depth parameter) or creates a new `GDevice` structure. Most constants used in creating a `GWorld` are irrelevant for this function, as its purpose is to wrap a `GWorld` around an existing block of pixels rather than to define and create a `pixmap`. See these constants:

baseAddr

The base address for the pixel data.

rowBytes

The total size of the pixel data divided by the height of the pixel map. In other words, the number of bytes in one row of pixels or the number of bytes between vertically adjacent pixels.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function wraps a `GWorld` around an existing pixel map. Note that it does not copy the `pixmap`. A subsequent call to `DisposeGWorld` will not dispose of the pixel map; it will only dispose of the `GWorld` wrapper. It is the caller's responsibility to dispose of the pixel map.

You can use this call to allocate an offscreen graphics world using special memory (such as on a video card). If you have an image in memory that belong to something else (a hardware screen buffer, a 3D card, or another file format or program), you can use this function to wrap a graphics world around the image and then use QuickTime calls on that graphics world to compress it, scale it, draw to it, and so on. If your new graphics world has a planar pixel format, you must use this call instead of `QTNewGWorld` (page 701).

Special Considerations

Do not unlock the pixels of the allocated graphics world. If your original pixels are from another graphics world then you must ensure that the source pixels are locked.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CTMClip

CTMDemo

OpenGLCompositorLab

OpenGLMovieQT

TextureRange

Declared In

`ImageCompression.h`

QTSetPixMapHandleGammaLevel

Sets the gamma level of a pixel map.

```
OSErr QTSetPixMapHandleGammaLevel (
   PixMapHandle pm,
   Fixed gammaLevel
);
```

Parameters

pm

A handle to a `PixMap` structure that has a `PixMapExtension` structure.

gammaLevel

The new gamma level.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function does not convert the contents of the `PixMap` structure. A typical usage would be to set the gamma level of a pixel map before compressing it so that the codec knows if it needs to do additional gamma correcting when compressing.

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

QTSetPixMapHandleRequestedGammaLevel

Sets the requested gamma level of a pixel map.

```
OSErr QTSetPixMapHandleRequestedGammaLevel (
   PixMapHandle pm,
   Fixed requestedGammaLevel
);
```

Parameters

pm

A handle to a `PixMap` structure that has a `PixMapExtension` structure.

requestedGammaLevel

A specified gamma level or a constant (see below). See these constants:

`kQTUsePlatformDefaultGammaLevel`

`kQTUseSourceGammaLevel`

`kQTCCIR601VideoGammaLevel`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function does not convert the contents of the `PixMap` structure. A typical usage would be to set the requested gamma level of a pixel map before decompressing so that the codec knows what gamma correction is necessary when decompressing into the `PixMap` structure. The resulting gamma level can then be found by calling `QTGetPixMapHandleGammaLevel` (page 698).

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

QTSetPixMapHandleRowBytes

Sets the rowBytes value for a pixel map accessed by a handle.

```
OSErr QTSetPixMapHandleRowBytes (
   PixMapHandle pm,
   long rowBytes
);
```

Parameters

pm

A handle to a `PixMap` structure.

rowBytes

The `rowBytes` value to be set.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

QTSetPixMapPtrGammaLevel

Sets the gamma level of a pixel map.

```
OSErr QTSetPixMapPtrGammaLevel (
   PixMapPtr pm,
   Fixed gammaLevel
);
```

Parameters

pm

A pointer to a `PixMap` structure that has a `PixMapExtension` structure.

gammaLevel

The new gamma level.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function does not convert the contents of the `PixMap` structure. A typical usage would be to set the gamma level of a pixel map before compressing it so that the codec knows if it needs to do additional gamma correcting when compressing.

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

QTSetPixMapPtrRequestedGammaLevel

Sets the requested gamma level of a pixel map.

```
OSErr QTSetPixMapPtrRequestedGammaLevel (
   PixMapPtr pm,
   Fixed requestedGammaLevel
);
```

Parameters

pm

A pointer to a `PixMap` structure that has a `PixMapExtension` structure.

requestedGammaLevel

A specified gamma level or a constant (see below). See these constants:

`kQTUsePlatformDefaultGammaLevel`

`kQTUseSourceGammaLevel`

`kQTCCIR601VideoGammaLevel`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function does not convert the contents of the `PixMap` structure. A typical usage would be to set the requested gamma level of a pixel map before decompressing so that the codec knows what gamma correction is necessary when decompressing into the `PixMap` structure. The resulting gamma level can then be found by calling `QTGetPixMapPtrGammaLevel` (page 699).

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

QTSetPixMapPtrRowBytes

Sets the rowBytes value for a pixel map accessed by a pointer.

```
OSErr QTSetPixMapPtrRowBytes (
   PixMapPtr pm,
   long rowBytes
);
```

Parameters

`pm`
A pointer to a `PixMap` structure.

`rowBytes`
The `rowBytes` value to be set.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

QTUpdateGWorld

Changes the pixel depth, boundary rectangle, or color table for an existing offscreen graphics world with a non-Macintosh pixel format.

```
GWorldFlags QTUpdateGWorld (
   GWorldPtr *offscreenGWorld,
   OSType PixelFormat,
   const Rect *boundsRect,
   CTabHandle cTable,
   GDHandle aGDevice,
   GWorldFlags flags
);
```

Parameters

`offscreenGWorld`
On input, a pointer to an existing offscreen graphics world; upon completion, the pointer to the updated offscreen graphics world.

`PixelFormat`
The updated graphics world's pixel format; see `Pixel Formats`.

boundsRect

A pointer to the boundary rectangle and port rectangle for the updated offscreen pixel map. This becomes the boundary rectangle for the `GDevice` structure, if this function creates one. If you specify 0 in the `PixelFormat` parameter, the function interprets the boundaries in global coordinates that it uses to determine which screens intersect the rectangle. It then uses the pixel format, color table, and `GDevice` structure from the screen with the greatest pixel depth from among all screens whose boundary rectangles intersect this rectangle. If the rectangle you specify in this parameter differs from, but has the same size as, the previous boundary rectangle, the function realigns the pixel image to the screen for optimum performance for `CopyBits`. Typically, your application supplies this parameter with the port rectangle for the onscreen window into which your application will copy the pixel image from this offscreen world.

cTable

A handle to a `ColorTable` structure for the updated graphics world. If you pass `NIL` in this parameter, the function uses the default color table for the pixel format that you specify in the `PixelFormat` parameter. If you set the `PixelFormat` parameter to 0, the function ignores the `cTable` parameter and instead copies and uses the color table of the graphics device with the greatest pixel depth among all graphics devices whose boundary rectangles intersect the rectangle that you specify in the `boundsRect` parameter. If the color table that you specify in this parameter is different from the previous color table, or if the color table associated with the `GDevice` structure that you specify in the `aGDevice` parameter is different, the function maps the pixel values in the offscreen pixel map to the new color table. If you use this function on a computer that supports only basic QuickDraw, you may specify only `NIL` in this parameter.

aGDevice

A handle to a `GDevice` structure that is used only when you specify the `noNewDevice` flag in the `flags` parameter, in which case the function attaches this structure to the new offscreen graphics world. If you set the `PixelFormat` parameter to 0, or if you do not set the `noNewDevice` flag, the function ignores this parameter, so you should set it to `NIL`. If you set the `PixelFormat` parameter to 0, the function uses the `GDevice` structure for the graphics device with the greatest pixel depth among all graphics devices whose boundary rectangles intersect the rectangle that you specify in the `boundsRect` parameter. You should pass `NIL` in this parameter if the computer supports only basic QuickDraw. Generally, your application should never create `GDevice` structures for offscreen graphics worlds.

flags

Constants (see below) that identify options available to your application. You can set a combination of these flags. If you don't wish to use any of them, pass 0 in this parameter. In this case the default behavior is to create an offscreen graphics world where the base address for the offscreen pixel image is unpurgeable, the graphics world uses an existing `GDevice` structure (if you pass 0 in the depth parameter) or creates a new `GDevice` structure, it uses memory in your application heap, and it allows graphics accelerators to cache the offscreen pixel image. See these constants:

Return Value

A constant (see below) that reports on the operation of this function.

Discussion

If the Memory Manager purged the base address for the offscreen pixel image, this function reallocates the memory but the pixel image is lost. You must reconstruct it.

Special Considerations

This function may move or purge memory blocks in the application heap. Your application should not call this function at interrupt time.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

QuadToQuadMatrix

Defines a matrix that maps between four input points and four output points.

```
OSErr QuadToQuadMatrix (
   const Fixed *source,
   const Fixed *dest,
   MatrixRecord *map
);
```

Parameters

source

A pointer to four input `FixedPoint` points.

dest

A pointer to four output `FixedPoint` points.

map

A pointer to a `MatrixRecord` structure that maps the value passed in `source` to the value passed in `dest`.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Graphic Import-Export

ImproveYourImage

Declared In
`ImageCompression.h`

RectMatrix

Creates a matrix that performs the translate and scale operation described by the relationship between two rectangles.

```
void RectMatrix (
   MatrixRecord *matrix,
   const Rect *srcRect,
   const Rect *dstRect
);
```

Parameters

matrix
: A pointer to a `MatrixRecord` structure. This function updates the contents of this matrix so that the matrix describes a transformation from points in the rectangle specified by the `srcRect` parameter to points in the rectangle specified by the `dstRect` parameter. The previous contents of the matrix are ignored.

srcRect
: A pointer to the source `Rect` structure.

dstRect
: A pointer to the destination `Rect` structure.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
AlwaysPreview
BrideOfMungGrab
JPEG File Interchange Format
SGDataProcSample
VideoProcessing

Declared In
`ImageCompression.h`

RemoveImageDescriptionExtension

Removes a specified extension from an ImageDescription structure.

```
OSErr RemoveImageDescriptionExtension (
   ImageDescriptionHandle desc,
   long idType,
   long index
);
```

Parameters

desc
: A handle to an `ImageDescription` structure.

idType
: The type of extension to remove.

index
: The index of the extension to remove. This is a number between 1 and the count returned by `CountImageDescriptionExtensionType` (page 618).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function allows an application to remove a specified extension from an `ImageDescription` structure. Note that any extensions that are present in the structure after the deleted extension will have their index numbers renumbered.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

ReplaceDSequenceImageDescription

Undocumented

```
OSErr ReplaceDSequenceImageDescription (
   ImageSequence seqID,
   ImageDescriptionHandle newDesc
);
```

Parameters

seqID
The unique sequence identifier assigned by `CompressSequenceBegin` (page 609) or `DecompressSequenceBegin` (page 621).

newDesc
A handle to an `ImageDescription` structure.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

RotateMatrix

Modifies the contents of a matrix so that it defines a rotation operation.

```
void RotateMatrix (
   MatrixRecord *m,
   Fixed degrees,
   Fixed aboutX,
   Fixed aboutY
);
```

Parameters

m

A pointer to a `MatrixRecord` structure.

degrees

The number of degrees of rotation.

aboutX

The x coordinate of the anchor point of rotation.

aboutY

The y coordinate of the anchor point of rotation.

Discussion

This function updates the contents of a matrix so that the matrix describes a rotation operation; that is, it concatenates the rotation transformations onto whatever was initially in the matrix structure. You specify the direction and amount of rotation with the `degrees` parameter. You specify the point of rotation with the `aboutX` and `aboutY` parameters.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

DropDraw

Graphic Import-Export

ImproveYourImage

qtgraphics

qtgraphics.win

Declared In

`ImageCompression.h`

ScaleMatrix

Modifies the contents of a matrix so that it defines a scaling operation.

```
void ScaleMatrix (
   MatrixRecord *m,
   Fixed scaleX,
   Fixed scaleY,
   Fixed aboutX,
   Fixed aboutY
);
```

Parameters

m

A pointer to a `MatrixRecord` structure. The `ScaleMatrix` function updates the contents of this matrix so that the matrix describes a scaling operation; that is, it concatenates the respective transformations onto whatever was initially in the matrix structure. You specify the magnitude of the scaling operation with the `scaleX` and `scaleY` parameters. You specify the anchor point with the `aboutX` and `aboutY` parameters.

scaleX

The scaling factor applied to x coordinates.

scaleY

The scaling factor applied to y coordinates.

aboutX

The x coordinate of the anchor point.

aboutY

The y coordinate of the anchor point.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CTMDemo

OpenGLMovieQT

qtgraphics

qtgraphics.win

TextureRange

Declared In

`ImageCompression.h`

SetCompressedPixMapInfo

Stores information about a compressed image for StdPix.

```
OSErr SetCompressedPixMapInfo (
   PixMapPtr pix,
   ImageDescriptionHandle desc,
   Ptr data,
   long bufferSize,
   ICMDataProcRecordPtr dataProc,
   ICMProgressProcRecordPtr progressProc
);
```

Parameters

pix

A pointer to a `PixMap` structure that holds compressed image data.

desc

A handle to the `ImageDescription` structure that defines the compressed image.

data

A pointer to the buffer for the compressed image data. If the entire compressed image cannot be stored at this location, you may assign a data-loading function (see the `dataProc` parameter, below). This pointer must contain a 32-bit clean address.

bufferSize

The size of the buffer to be used by the data-loading function specified by the `dataProc` parameter. If there is no data-loading function defined for this operation, set this parameter to 0.

dataProc

A pointer to an `ICMDataProcRecord` structure. If there is not enough memory to store the compressed image, the decompressor calls an `ICMDataProc` callback that you provide, which loads more compressed data. If you do not want to assign a data-loading function, set this parameter to `NIL`.

progressProc

A pointer to an `ICMProgressProcRecord` structure. During the decompression operation, the decompressor may occasionally call an `ICMProgressProc` callback that you provide, in order to report its progress. If you do not want to assign a progress function, set this parameter to `NIL`. If you pass a value of -1, you obtain a standard progress function.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

SetCSequenceDataRateParams

Communicates information to compressors that can constrain compressed data in a particular sequence to a specific data rate.

```
OSErr SetCSequenceDataRateParams (
   ImageSequence seqID,
   DataRateParamsPtr params
);
```

Parameters

seqID

The unique sequence identifier assigned by CompressSequenceBegin (page 609) or DecompressSequenceBegin (page 621).

params

A pointer to a DataRateParams structure.

Return Value

See Error Codes. Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

ImageCompression.h

SetCSequenceFlushProc

Assigns a data-unloading function to a sequence.

```
OSErr SetCSequenceFlushProc (
   ImageSequence seqID,
   ICMFlushProcRecordPtr flushProc,
   long bufferSize
);
```

Parameters

seqID

The unique sequence identifier assigned by CompressSequenceBegin (page 609) or DecompressSequenceBegin (page 621).

flushProc

A pointer to an ICMFlushProcRecord structure. If there is not enough memory to store the compressed image, the compressor calls an ICMFlushProc callback that you provide, which unloads some of the compressed data. If you have not provided such a data-unloading function, set this parameter to NIL. In this case, the compressor writes the entire compressed image into the memory location specified by the data parameter to CompressSequenceFrame (page 612).

bufferSize

The size of the buffer to be used by the data-unloading function specified by the flushProc parameter. If you have not specified such a data-unloading function, set this parameter to 0.

Return Value

See Error Codes. Returns noErr if there is no error.

Discussion

Data-unloading functions allow compressors to work with images that cannot fit in memory. During the compression operation, the compressor calls the data-unloading function whenever it has accumulated a specified amount of compressed data. Your data-unloading function then writes the compressed data to some other device, freeing buffer space for more compressed data. The compressor starts using the data-unloading function with the next image in the sequence.

Special Considerations

There is no parameter to the `CompressSequenceBegin` (page 609) function that allows you to assign a data-unloading function to a sequence.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

SetCSequenceFrameNumber

Informs the compressor in use for the specified sequence that frames are being compressed out of order.

```
OSErr SetCSequenceFrameNumber (
   ImageSequence seqID,
   long frameNumber
);
```

Parameters

seqID

The unique sequence identifier assigned by `CompressSequenceBegin` (page 609) or `DecompressSequenceBegin` (page 621).

frameNumber

The frame number of the frame that is being compressed out of sequence.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This information is necessary only for compressors that are sequence-sensitive.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

SetCSequenceKeyFrameRate

Adjusts the key frame rate for the current sequence.

```
OSErr SetCSequenceKeyFrameRate (
   ImageSequence seqID,
   long keyFrameRate
);
```

Parameters

seqID

The unique sequence identifier assigned by `CompressSequenceBegin` (page 609) or `DecompressSequenceBegin` (page 621).

keyFrameRate

The maximum number of frames allowed between key frames. Set this parameter to 1 to specify all key frames, to 2 to specify every other frame as a key frame, to 3 to specify every third frame as a key frame, and so forth. The compressor determines the optimum placement for key frames based upon the amount of redundancy between adjacent images in the sequence. Consequently, the compressor may insert key frames more frequently than you have requested. However, the compressor will never place fewer key frames than is indicated by this parameter. If you set this parameter to 0, the Image Compression Manager only places key frames in the compressed sequence when you call `CompressSequenceFrame` (page 612), setting the `codecFlagForceKeyFrame` flag in the `flags` parameter. The compressor ignores this parameter if you have not requested temporal compression; that is, you have passed 0 for the `temporalQuality` parameter of `CompressSequenceBegin` (page 609).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Key frames provide points from which a temporally compressed sequence may be decompressed. Use this parameter to control the frequency at which the compressor places key frames into the compressed sequence. The new key frame rate takes effect with the next image in the sequence.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

SetCSequencePreferredPacketSize

Sets the preferred packet size for a sequence.

```
OSErr SetCSequencePreferredPacketSize (
   ImageSequence seqID,
   long preferredPacketSizeInBytes
);
```

Parameters

seqID

The unique sequence identifier assigned by `CompressSequenceBegin` (page 609) or `DecompressSequenceBegin` (page 621).

preferredPacketSizeInBytes

The preferred packet size in bytes.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

This function was added in QuickTime 2.5 to support video conferencing applications by making each transmitted packet an independently decodable chunk of data.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

SetCSequencePrev

Allows the application to set the pixel map and boundary rectangle used by the previous frame in temporal compression.

```
OSErr SetCSequencePrev (
   ImageSequence seqID,
   PixMapHandle prev,
   const Rect *prevRect
);
```

Parameters

seqID

The unique sequence identifier assigned by `CompressSequenceBegin` (page 609) or `DecompressSequenceBegin` (page 621).

prev

A handle to the new previous image buffer. You must allocate this buffer using the same pixel depth and `ColorTable` structure as the source image buffer that you specified with the `src` parameter when you called `CompressSequenceBegin` (page 609). The compressor uses this buffer to store a previous image against which the current image is compared when performing temporal compression. The compressor manages the contents of this buffer based upon several considerations, such as the key frame rate and the degree of difference between compared images. The current image is stored in the buffer referred to by the `src` parameter to `CompressSequenceBegin`.

prevRect

A pointer to a `Rect` structure that defines the portion of the previous image to use for temporal compression. The compressor uses this portion of the previous image as the basis of comparison with the current image. This rectangle must be the same size as the source rectangle you specify with the `srcRect` parameter to `CompressSequenceBegin` (page 609). To get the boundary of a source pixel map, set this parameter to `NIL`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

When you start compressing a sequence, you may assign a previous frame buffer and rectangle with the `prev` and `prevRect` parameters to `CompressSequenceBegin` (page 609). If you specified a `NIL` value for the `prev` parameter, the compressor allocates an offscreen buffer for the previous frame. In either case you may use this function to assign a new previous frame buffer.

Special Considerations

This is a very specialized function; your application should not need to call it under most circumstances.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Desktop Sprites

qteffects.win

qtsprites.win

qtwiredactions

qtwiredsprites

Declared In

`ImageCompression.h`

SetCSequenceQuality

Adjusts the spatial or temporal quality for the current sequence.

```
OSErr SetCSequenceQuality (
   ImageSequence seqID,
   CodecQ spatialQuality,
   CodecQ temporalQuality
);
```

Parameters

seqID

The unique sequence identifier assigned by `CompressSequenceBegin` (page 609) or `DecompressSequenceBegin` (page 621).

spatialQuality

A constant (see below) that specifies the desired compressed image quality. See these constants:

`codecMinQuality`

`codecLowQuality`

`codecNormalQuality`

`codecHighQuality`

`codecMaxQuality`

`codecLosslessQuality`

temporalQuality

A constant (see below) that specifies the desired sequence temporal quality. This parameter governs the level of compression you desire with respect to information between successive frames in the sequence. Set this parameter to 0 to prevent the compressor from applying temporal compression to the sequence.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

You originally set the default spatial and temporal quality values for a sequence with `CompressSequenceBegin` (page 609). The new quality parameters take effect with the next frame in the sequence.

Special Considerations

If you change the quality settings while processing an image sequence, you affect the maximum image size that you may receive during sequence compression. Consequently, you should call `GetMaxCompressionSize` (page 670) after you change the quality settings. If the maximum size has increased, you should reallocate your image buffers to accommodate the larger image size.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

SetDSequenceAccuracy

Adjusts the decompression accuracy for the current sequence.

```
OSErr SetDSequenceAccuracy (
   ImageSequence seqID,
   CodecQ accuracy
);
```

Parameters

seqID

The unique sequence identifier assigned by `CompressSequenceBegin` (page 609) or `DecompressSequenceBegin` (page 621).

accuracy

A constant (see below) that specifies the accuracy desired in the decompressed image. See these constants:

`codecMinQuality`

`codecLowQuality`

`codecNormalQuality`

`codecHighQuality`

`codecMaxQuality`

`codecLosslessQuality`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The accuracy parameter governs how precisely the decompressor decompresses the image data. Some decompressors may choose to ignore some image data to improve decompression speed. A new accuracy value takes effect with the next frame in the sequence.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

SetDSequenceDataProc

Assigns a data-loading function to a sequence.

```
OSErr SetDSequenceDataProc (
   ImageSequence seqID,
   ICMDataProcRecordPtr dataProc,
   long bufferSize
);
```

Parameters

seqID

The unique sequence identifier assigned by `CompressSequenceBegin` (page 609) or `DecompressSequenceBegin` (page 621).

dataProc

A pointer to an `ICMDataProcRecord` structure. If the data stream is not all in memory when your program calls `DecompressSequenceFrame` (page 624), the decompressor calls an `ICMDataProc` callback that you provide, which loads more compressed data. If you have not provided such a data-loading function, or if you want the decompressor to stop using your data-loading function, set this parameter to `NIL`. In this case, the entire image must be in memory at the location specified by the `data` parameter to `DecompressSequenceFrame`.

bufferSize

The size of the buffer to be used by the data-loading function specified by the `dataProc` parameter. If you have not specified a data-loading function, set this parameter to 0.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Data-loading functions allow decompressors to work with images that cannot fit in memory. During the decompression operation the decompressor calls the data-loading function whenever it has exhausted its supply of compressed data.

Special Considerations

There is no parameter to the `DecompressSequenceBegin` (page 621) function that allows you to assign a data-loading function to a sequence.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

SetDSequenceFlags

Sets data loading flags.

```
OSErr SetDSequenceFlags (
   ImageSequence seqID,
   long flags,
   long flagsMask
);
```

Parameters

seqID

The unique sequence identifier assigned by `CompressSequenceBegin` (page 609) or `DecompressSequenceBegin` (page 621).

flags

Flags (see below) for data loading. See these constants:

`codecDSequenceSingleField`

flagsMask

Use this field to preserve the state of any flags you do not wish to alter. If a flag (see below) is set in this field, and is not set in the `flags` parameter, it will not be changed from its current setting.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

MungSaver

Declared In

`ImageCompression.h`

SetDSequenceMask

Assigns a clipping region to a sequence.

```
OSErr SetDSequenceMask (
   ImageSequence seqID,
   RgnHandle mask
);
```

Parameters

seqID

The unique sequence identifier assigned by `CompressSequenceBegin` (page 609) or `DecompressSequenceBegin` (page 621).

mask

A handle to a clipping region in the destination coordinate system. If specified, the decompressor applies this mask to the destination image. If you want to stop masking, set this parameter to `NIL`. The new region takes effect with the next frame in the sequence.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The decompressor draws only that portion of the decompressed image that lies within the specified clipping region. You should not dispose of this region until the Image Compression Manager is finished with the sequence, or until you set the mask either to `NIL` or to a different region by calling this function again.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

SetDSequenceMatrix

Assigns a mapping matrix to a sequence.

```
OSErr SetDSequenceMatrix (
   ImageSequence seqID,
   MatrixRecordPtr matrix
);
```

Parameters

seqID

The unique sequence identifier assigned by `CompressSequenceBegin` (page 609) or `DecompressSequenceBegin` (page 621).

matrix

A `MatrixRecord` structure that specifies how to transform the image during decompression. You can use this structure to translate or scale the image during decompression. To set the matrix to identity, pass `NIL` in this parameter. The new matrix takes effect with the next frame in the sequence.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The decompressor uses the matrix to create special effects with the decompressed image, such as translating or scaling the image.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

SetDSequenceMatte

Assigns a blend matte to a sequence.

```
OSErr SetDSequenceMatte (
   ImageSequence seqID,
   PixMapHandle matte,
   const Rect *matteRect
);
```

Parameters

seqID

The unique sequence identifier assigned by `CompressSequenceBegin` (page 609) or `DecompressSequenceBegin` (page 621).

matte

A handle to a `PixMap` structure that contains a blend matte. You can use the blend matte to cause the decompressed image to be blended into the destination pixel map. The matte can be defined at any supported pixel depth; the matte depth need not correspond to the source or destination depths. However, the matte must be in the coordinate system of the source image. If you want to turn off the blend matte, set this parameter to `NIL`.

matteRect

A pointer to a `Rect` structure that defines the boundary rectangle for the matte. The decompressor uses only that portion of the matte that lies within the specified rectangle. This rectangle must be the same size as the source rectangle you specify with `SetDSequenceSrcRect` (page 727) or with the `srcRect` parameter to `DecompressSequenceBegin` (page 621). To use the matte's `PixMap` structure bounds as the boundary rectangle, pass `NIL` in this parameter.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion
The decompressor uses the matte to blend the decompressed image into the destination pixel map. The new matte and matte boundary rectangle take effect with the next frame in the sequence. You should not dispose of the matte until the Image Compression Manager has finished with the sequence.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.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` (page 621) 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.

Availability
Available in Mac OS X v10.3 and later.

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` (page 621) 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.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`ImageCompression.h`

SetDSequenceSrcRect

Defines the portion of an image to decompress.

```
OSErr SetDSequenceSrcRect (
   ImageSequence seqID,
   const Rect *srcRect
);
```

Parameters

seqID
: The unique sequence identifier assigned by `CompressSequenceBegin` (page 609) or `DecompressSequenceBegin` (page 621).

srcRect
: A pointer to a `Rect` structure that defines the portion of the image to decompress. This rectangle must lie within the boundary rectangle of the compressed image, which is defined by (0,0) and `((**desc).width,(**desc).height)`, where `desc` refers to the `ImageDescription` structure you supply to `DecompressSequenceBegin` (page 621). If the `srcRect` parameter is `NIL`, the rectangle is set to the `Rect` structure in the `ImageDescription`.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
The decompressor acts on that portion of the compressed image that lies within this rectangle. A new source rectangle takes effect with the next frame in the sequence.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

SetDSequenceTimeCode

Sets the timecode value for a frame that is about to be decompressed.

```
OSErr SetDSequenceTimeCode (
   ImageSequence seqID,
   void *timeCodeFormat,
   void *timeCodeTime
);
```

Parameters

seqID
The unique sequence identifier assigned by `CompressSequenceBegin` (page 609) or `DecompressSequenceBegin` (page 621).

timeCodeFormat
A pointer to a `TimeCodeDef` structure. You provide the appropriate timecode definition information for the next frame to be decompressed.

timeCodeTime
A pointer to a `TimeCodeRecord` structure. You provide the appropriate time value for the next frame in the current sequence.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
QuickTime's video media handler uses this function to set the timecode information for a movie. When a movie that contains timecode information starts playing, the media handler calls this function as it processes the movie's first frame. The Image Compression Manager passes the timecode information straight through to the image decompressor component. That is, the Image Compression Manager does not make a copy of any of this timecode information. As a result, you must make sure that the data referred to by the `timeCodeFormat` and `timeCodeTime` parameters is valid until the next decompression operation completes.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

SetDSequenceTransferMode

Sets the mode used when drawing a decompressed image.

```
OSErr SetDSequenceTransferMode (
   ImageSequence seqID,
   short mode,
   const RGBColor *opColor
);
```

Parameters

seqID

The unique sequence identifier assigned by CompressSequenceBegin (page 609) or DecompressSequenceBegin (page 621).

mode

A constant (see below) that specifies the transfer mode to be used when drawing the decompressed image. See also Graphics Transfer Modes. See these constants:

opColor

Contains a pointer to the color for use in addPin, subPin, blend, and transparent operations. The Image Compression Manager passes this color to QuickDraw as appropriate. If NIL, the opcolor is left unchanged.

Return Value

See Error Codes. Returns noErr if there is no error.

Discussion

For any given sequence, the default opColor value is 50 percent gray and the default mode is ditherCopy. The new mode takes effect with the next frame in the sequence.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

ImageCompression.h

SetIdentityMatrix

Sets the contents of a matrix so that it performs no transformation.

```
void SetIdentityMatrix (
   MatrixRecord *matrix
);
```

Parameters

matrix

A pointer to a MatrixRecord structure. The function updates the contents of this matrix so that the matrix describes the identity matrix.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Graphic Import-Export

ImproveYourImage

qtspritesplus.win

vrmovies

vrmovies.win

Declared In

`ImageCompression.h`

SetImageDescriptionCTable

Updates the custom ColorTable structure for an image.

```
OSErr SetImageDescriptionCTable (
   ImageDescriptionHandle desc,
   CTabHandle ctable
);
```

Parameters

`desc`

Contains a handle to the appropriate `ImageDescription` structure. The function updates the size of the structure to accommodate the new `ColorTable` structure and removes the old color table, if one is present.

`ctable`

A handle to the new `ColorTable` structure. The function loads this color table into the `ImageDescription` structure referred to by the `desc` parameter. Set this parameter to `NIL` to remove a `ColorTable` structure.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function does not change the image data, just the color table.

Special Considerations

This function is rarely used. Typically, you supply the color table when your application compresses an image, and the Image Compression Manager stores the `ColorTable` structure with the image.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Desktop Sprites

qteffects.win

qtsprites.win

qtwiredactions

qtwiredsprites

Declared In
`ImageCompression.h`

SetSequenceProgressProc

Installs a progress procedure for a sequence.

```
OSErr SetSequenceProgressProc (
   ImageSequence seqID,
   ICMProgressProcRecord *progressProc
);
```

Parameters

`seqID`
: The unique sequence identifier assigned by `CompressSequenceBegin` (page 609) or `DecompressSequenceBegin` (page 621).

`progressProc`
: A pointer to an `ICMProgressProcRecord` structure.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function allows you to set an `ICMProgressProc` callback for a compression or decompression sequence, just as you can set a progress procedure when compressing or decompressing a still image.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

SkewMatrix

Modifies the contents of a matrix so that it defines a skew transformation.

```
void SkewMatrix (
   MatrixRecord *m,
   Fixed skewX,
   Fixed skewY,
   Fixed aboutX,
   Fixed aboutY
);
```

Parameters

m

A pointer to the matrix for this operation. The `SkewMatrix` function updates the contents of the `MatrixRecord` structure so that it defines a skew operation; it concatenates the respective transformations onto whatever was initially in the matrix structure. You specify the magnitude and direction of the skew operation with the `skewX` and `skewY` parameters. You specify an anchor point with the `aboutX` and `aboutY` parameters.

skewX

The skew value to be applied to x coordinates.

skewY

The skew value to be applied to y coordinates.

aboutX

The x coordinate of the anchor point.

aboutY

The y coordinate of the anchor point.

Discussion

A skew operation alters the display of an element along one dimension. For example, converting a rectangle into a parallelogram is a skew operation.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

StdPix

Extends the grafProcs field of the CGrafPort structure to support compressed data, mattes, matrices, and pixel maps, letting you intercept image data in compressed form before it is decompressed and displayed.

```
void StdPix (
   PixMapPtr src,
   const Rect *srcRect,
   MatrixRecordPtr matrix,
   short mode,
   RgnHandle mask,
   PixMapPtr matte,
   const Rect *matteRect,
   short flags
);
```

Parameters

src

Contains a pointer to a PixMap structure containing the image to draw. Use GetCompressedPixMapInfo (page 654) to retrieve information about this structure.

srcRect

Points to a Rect structure that defines the portion of the image to display. This rectangle must lie within the boundary rectangle of the compressed image or within the source image. If this parameter is set to NIL, the entire image is displayed.

matrix

Contains a pointer to a MatrixRecord structure that specifies the mapping of the source rectangle to the destination. It is a fixed-point, 3-by-3 matrix.

mode

Specifies the transfer mode for the operation; see Graphics Transfer Modes. Note that this parameter also controls the accuracy of any decompression operation that may be required to display the image. If bit 7 (0x80) of the mode parameter is set to 1, the StdPix function sets the decompression accuracy to codecNormalQuality. If this bit is set to 0, the function sets the accuracy to codecHighQuality.

mask

Contains a handle to a clipping region in the destination coordinate system. If specified, the compressor applies this mask to the destination image. If there is no mask, this parameter is set to NIL.

matte

Points to a PixMap structure that contains a blend matte. The blend matte causes the decompressed image to be blended into the destination pixel map. The matte can be defined at any supported pixel depth; the matte depth need not correspond to the source or destination depths. However, the matte must be in the coordinate system of the source image. If there is no matte, this parameter is set to NIL

matteRect

Contains a pointer to a Rect structure that defines a portion of the blend matte to apply. This parameter is set to NIL if there is no matte or if the entire matte is to be used.

flags

Contains control flags (see below). See these constants:

callOldBits

callStdBits

noDefaultOpcodes

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

TransformFixedPoints

Transforms a set of fixed points through a specified matrix.

```
OSErr TransformFixedPoints (
   const MatrixRecord *m,
   FixedPoint *fpt,
   long count
);
```

Parameters

m
A pointer to the transformation matrix for this operation.

fpt
A pointer to the first fixed point to be transformed.

count
The number of fixed points to be transformed. These points must be stored immediately following the point specified by the `fpt` parameter.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

TransformFixedRect

Transforms the upper-left and lower-right points of a rectangle through a matrix that is specified by fixed points.

```
Boolean TransformFixedRect (
   const MatrixRecord *m,
   FixedRect *fr,
   FixedPoint *fpp
);
```

Parameters

m
A pointer to the matrix for this operation.

fr

A pointer to the `FixedRect` structure that defines the rectangle to be transformed. `TransformFixedRect` returns the updated coordinates into the structure referred to by this parameter. If the resulting rectangle has been rotated or skewed (that is, the transformation involves operations other than scaling and translation), the function sets the returned Boolean value to FALSE and returns the coordinates of the boundary box of the transformed rectangle. The function then updates the points specified by the `fpp` parameter to contain the coordinates of the four corners of the transformed rectangle.

fpp

A pointer to an array of four fixed points. The function returns the coordinates of the four corners of the rectangle after the transformation operation. If you do not want this information, set this parameter to `NIL`.

Return Value

If the resulting rectangle has been rotated or skewed (that is, the transformation involves operations other than scaling and translation), the function returns FALSE, updates the rectangle specified by the `fr` parameter to define the boundary box of the resulting rectangle, and places the coordinates of the corners of the resulting rectangle in the points specified by the `fpp` parameter. If the transformed rectangle and its boundary box are the same, the function returns TRUE.

Discussion

This function does not return any error codes.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

TransformPoints

Transforms a set of QuickDraw points through a specified matrix.

```
OSErr TransformPoints (
   const MatrixRecord *mp,
   Point *pt1,
   long count
);
```

Parameters

mp

A pointer to the transformation matrix for this operation.

pt1

A pointer to the first QuickDraw point to be transformed.

count

The number of QuickDraw points to be transformed. These points must be stored immediately following the point specified by the pt1 parameter.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

TransformRect

Transforms the upper-left and lower-right points of a rectangle through a specified matrix.

```
Boolean TransformRect (
   const MatrixRecord *m,
   Rect *r,
   FixedPoint *fpp
);
```

Parameters

m

The matrix for this operation.

r

A pointer to the `Rect` structure that defines the rectangle to be transformed. The function returns the updated coordinates into the structure referred to by this parameter.

fpp

A pointer to an array of four fixed points. The `TransformRect` function returns the coordinates of the four corners of the rectangle after the transformation operation. If you do not want this information, set this parameter to `NIL`.

Return Value
If the resulting rectangle has been rotated or skewed (that is, the transformation involves operations other than scaling and translation), the function returns FALSE, updates the rectangle specified by the `r` parameter to define the boundary box of the resulting rectangle, and places the coordinates of the corners of the resulting rectangle in the points specified by the `fpp` parameter. If the transformed rectangle and its boundary box are the same, the function returns TRUE.

Discussion
This function does not return any error codes.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
AlwaysPreview

BurntTextSampleCode

DrawTextCodec

ExampleCodec

qdmediahandler.win

Declared In

`ImageCompression.h`

TransformRgn

Applies a specified matrix to a region.

```
OSErr TransformRgn (
   MatrixRecordPtr matrix,
   RgnHandle rgn
);
```

Parameters

matrix
: Points to the matrix for this operation. The `TransformRgn` function currently supports only translation and scaling operations.

rgn
: A handle to the `MacRegion` structure to be transformed. The function transforms each point in the region according to the specified matrix

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qtskins

qtskins.win

Declared In

`ImageCompression.h`

TranslateMatrix

Adds a translation value to a specified matrix.

```
void TranslateMatrix (
   MatrixRecord *m,
   Fixed deltaH,
   Fixed deltaV
);
```

Parameters

m
: A pointer to the `MatrixRecord` structure for this operation.

deltaH
: The value to be added to the x coordinate translation value.

deltaV

The value to be added to the y coordinate translation value.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

GroupDrawing

ImageCompositing

qtgraphics

qtgraphics.win

qtspritesplus.win

Declared In

`ImageCompression.h`

TrimImage

Adjusts a compressed image to the boundaries defined by a specified rectangle.

```
ComponentResult ADD_IMAGECODEC_BASENAME() TrimImage
```

Parameters

desc

A handle to the `ImageDescription` structure that describes the compressed image. On return, the compressor updates this structure to describe the resized image.

inData

A pointer to the compressed image data. This pointer must contain a 32-bit clean address. If the entire compressed image cannot be stored at this location, your application may provide an `ICMDataProc` callback through the `dataProc` parameter.

inBufferSize

The size of the buffer to be used by the data-loading function specified by the `dataProc` parameter. If you have not specified a data-loading function, this parameter is ignored.

dataProc

A pointer to an `ICMDataProcRecord` structure that references an `ICMDataProc` callback. If there is not enough memory to store the compressed image, the compressor calls a function you provide that loads more compressed data. If you have not provided such a data-loading function, set this parameter to `NIL`. In this case, the compressor expects that the entire compressed image is in the memory location specified by the `inData` parameter

outData

A pointer to a buffer to receive the trimmed image. The Image Compression Manager places the actual size of the resulting image into the `dataSize` field of the `ImageDescription` structure referred to by the `desc` parameter. This pointer must contain a 32-bit clean address. Your application should create a destination buffer at least as large as the source image. If there is not sufficient memory to store the compressed image, you may choose to write the compressed data to mass storage during the compression operation, in which case you use the `flushProc` parameter to identify your data-unloading function to the compressor.

outBufferSize

The size of the buffer to be used by the data-unloading function specified by the `flushProc` parameter. If you have not specified a data-unloading function, this parameter is ignored.

flushProc

A pointer to an `ICMFlushProcRecord` structure that references an `ICMFlushProc` callback. If there is not enough memory to store the compressed image, the compressor calls a function you provide that unloads some of the compressed data. If you have not provided such a data-unloading function, set this parameter to `NIL`. In this case, the compressor writes the entire compressed image into the memory location specified by the `data` parameter

trimRect

A pointer to a `Rect` structure that defines the desired image dimensions. On return, the function adjusts the rectangle values so that they refer to the same rectangle in the result image. This is necessary whenever data is removed from the beginning or left side of the image.

progressProc

A pointer to an `ICMProgressProcRecord` structure that references an `ICMProgressProc` callback. During the operation, the compressor may occasionally call a function you provide in order to report its progress. If you have not provided such a progress function, set this parameter to `NIL`. If you pass a value of -1, you obtain a standard progress function.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

UnsignedFixMulDiv

Performs multiplications and divisions on unsigned fixed-point numbers.

```
Fixed UnsignedFixMulDiv (
   Fixed src,
   Fixed mul,
   Fixed divisor
);
```

Parameters

src

The value to be multiplied or divided.

mul

The multiplier to be applied to the value in the `src` parameter. Pass 0x00010000 if you do not want to multiply.

divisor

The divisor to be applied to the value in the `src` parameter. Pass 0x00010000 if you do not want to divide.

Return Value

The fixed-point number that is the value of the `src` parameter, multiplied by the value in the `mul` parameter and divided by the value in the `divisor` parameter. The function performs both operations before returning.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

Callbacks

Data Types

CodecComponent

Represents a type used by the Compression and Decompression API.

```
typedef Component CodecComponent;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

CodecNameSpecList

Contains a list of CodecNameSpec structures.

```
struct CodecNameSpecList {
     short            count;
     CodecNameSpec    list[1];
 };
```

Fields

`count`

Discussion

Indicates the number of compressor name structures contained in the list array that follows.

`list`

Discussion

Contains an array of compressor name structures. Each structure corresponds to one compressor component or type that meets the selection criteria your application specifies when it calls `GetCodecNameList` (page 652).

Related Functions
`DisposeCodecNameList` (page 628)
`GetCodecNameList` (page 652)

Declared In
`ImageCompression.h`

CodecNameSpecListPtr

Represents a type used by the Compression and Decompression API.

```
typedef CodecNameSpecList * CodecNameSpecListPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

ConstStrFileNameParam

Represents a type used by the Compression and Decompression API.

```
typedef ConstStr255Param ConstStrFileNameParam;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacTypes.h`

DataRateParams

Communicates information to compressors that can constrain compressed data to a specific data rate.

```
struct DataRateParams {
     long      dataRate;
     long      dataOverrun;
     long      frameDuration;
     long      keyFrameRate;
     CodecQ    minSpatialQuality;
     CodecQ    minTemporalQuality;
 };
```

Fields
`dataRate`
Discussion
Specifies the bytes per second to which the data rate must be constrained.

`dataOverrun`
Discussion
Indicates the current number of bytes above or below the desired data rate. A value of 0 means that the data rate is being met exactly. If your application doesn't know the data overrun, it should set this field to 0.

`frameDuration`

Discussion
Specifies the duration of the current frame in milliseconds.

`keyFrameRate`

Discussion
Indicates the frequency of key frames. This frequency is normally identical to the key frame rate passed to the `CompressSequenceBegin` (page 609).

`minSpatialQuality`

Discussion
A constant (see below) that specifies the minimum spatial quality the compressor should use to meet the requested data rate. See these constants:

    codecMinQuality
    codecLowQuality
    codecNormalQuality
    codecHighQuality
    codecMaxQuality
    codecLosslessQuality

`minTemporalQuality`

Discussion
A constant (see below) that specifies the minimum temporal quality the compressor should use to meet the requested data rate.

Discussion
The `CodecQ` data type defines a field that identifies the quality characteristics of a given image or sequence. Note that individual components may not implement all the quality levels shown here. In addition, components may implement other quality levels in the range from `codecMinQuality` to `codecMaxQuality`. Relative quality should scale within the defined value range. Values above `codecLosslessQuality` are reserved for use by individual components.

Related Functions
`GetCSequenceDataRateParams` (page 657)
`SetCSequenceDataRateParams` (page 715)

Declared In
`ImageCompression.h`

DataRateParamsPtr

Represents a type used by the Compression and Decompression API.

```
typedef DataRateParams * DataRateParamsPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

DecompressorComponent

Represents a type used by the Compression and Decompression API.

```
typedef Component DecompressorComponent;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

FixedRect

Defines the size and location of a rectangle in fixed-point numbers.

```
struct FixedRect {
     Fixed    left;
     Fixed    top;
     Fixed    right;
     Fixed    bottom;
 };
```

Fields
`left`

Discussion
The x coordinate of the upper-left corner of the rectangle.

`top`

Discussion
The y coordinate of the upper-left corner of the rectangle.

`right`

Discussion
The x coordinate of the lower-right corner of the rectangle.

`bottom`

Discussion
The y coordinate of the lower-right corner of the rectangle.

Related Functions
`TransformFixedRect` (page 734)

Declared In
`ImageCompression.h`

Fract

Represents a type used by the Compression and Decompression API.

```
typedef long Fract;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`IOMacOSTypes.h`

ICMPixelFormatInfo

Defines a pixel format.

```
struct ICMPixelFormatInfo {
     long             size;
     unsigned long    formatFlags;
     short            bitsPerPixel[14];
 };
```

Fields

`size`

Discussion
The size of this structure. This quantity isn't necessarily equal to `sizeof(ICMPixelFormatInfo)` because it is dependent on whether the pixel format is chunky or planar, and, if planar, the number of components (see below).

`formatFlags`

Discussion
A constant (see below) indicating the pixel format. See these constants:

`kICMPixelFormatIsPlanarMask`

`kICMPixelFormatIsIndexed`

`kICMPixelFormatIsSupportedByQD`

`bitsPerPixel`

Discussion
An array that defines the number of bits for each component. The element `bitsPerPixel[0]` contains the number of bits for the first component, `bitsPerPixel[1]` the number of bits for the second component, etc. The meaning of this parameter depends on the format flag (see below).

Discussion
You can represent a format that has from 1 to 14 discrete components using this data structure. For ARGB, there are 4 components. RGB without an alpha channel has 3 components. A component count of 15 is reserved for future expansion.

Related Functions
`ICMGetPixelFormatInfo` (page 677)
`ICMSetPixelFormatInfo` (page 681)

Declared In
`ImageCompression.h`

ICMPixelFormatInfoPtr

Represents a type used by the Compression and Decompression API.

```
typedef ICMPixelFormatInfo * ICMPixelFormatInfoPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

ImageFieldSequence

Represents a type used by the Compression and Decompression API.

```
typedef long ImageFieldSequence;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

ImageSequenceDataSource

Represents a type used by the Compression and Decompression API.

```
typedef long ImageSequenceDataSource;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

ImageTranscodeSequence

Represents a type used by the Compression and Decompression API.

```
typedef long ImageTranscodeSequence;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

OpenCPicParams

Specifies resolutions when creating images.

```
struct OpenCPicParams {
     Rect     srcRect;
     Fixed    hRes;
     Fixed    vRes;
     short    version;
     short    reserved1;
     long     reserved2;
 };
```

Fields

`srcRect`

Discussion

The optimal bounding rectangle for the resolution indicated by the `hRes` and `vRes` fields. To display a picture at a resolution other than that specified in the the `hRes` and `vRes` fields, your application should compute an appropriate destination rectangle by scaling the image's width and height by the destination resolution divided by the source resolution.

`hRes`

Discussion

The best horizontal resolution for the picture. A value of 0x0048000 specifies a horizontal resolution of 72 dpi.

`vRes`

Discussion

The best vertical resolution for the picture. A value of 0x0048000 specifies a vertical resolution of 72 dpi.

`version`

Discussion

Always set this field to -2.

`reserved1`

Discussion

Reserved; set to 0.

`reserved2`

Discussion

Reserved; set to 0.

Related Functions

`GetPictureFileHeader` (page 673)

Declared In

`ImageCompression.h`

QHdr

A Windows queue header structure.

```
struct QHdr {
     short       qFlags;
     short       pad;
     long        MutexID;
     QElemPtr    qHead;
     QElemPtr    qTail;
 };
```

Fields

`qFlags`

Discussion

Undocumented

`pad`

Discussion

Unused.

`MutexID`

Discussion

Undocumented

`qHead`

Discussion

Undocumented

`qTail`

Discussion

Undocumented

Related Functions

`CDSequenceSetSourceDataQueue` (page 599)
`Dequeue`
`Enqueue`
`InitializeQHdr`
`TerminateQHdr`

Declared In

`ImageCompression.h`

QHdrPtr

Represents a type used by the Compression and Decompression API.

```
typedef QHdr * QHdrPtr;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OSUtils.h`

Constants

StdPix Values

Constants passed to StdPix.

```
enum {
  callStdBits                  = 1,
  callOldBits                  = 2,
  noDefaultOpcodes             = 4
};
```

Declared In
ImageCompression.h

DSequence Flags

Constants that represent <codeVoice>DSequence</codeVoice> flags.

```
enum {
  codecDSequenceDisableOverlaySurface = (1L << 5),
  codecDSequenceSingleField     = (1L << 6),
  codecDSequenceBidirectionalPrediction = (1L << 7),
  codecDSequenceFlushInsteadOfDirtying = (1L << 8),
  codecDSequenceEnableSubPixelPositioning = (1L << 9),
  codecDSequenceDeinterlaceFields = (1L << 10)
};
```

Declared In
ImageCompression.h

FCompressImage Values

Constants passed to FCompressImage.

```
enum {
  codecFlagUseImageBuffer       = (1L << 0), /* decompress*/
  codecFlagUseScreenBuffer      = (1L << 1), /* decompress*/
  codecFlagUpdatePrevious       = (1L << 2), /* compress*/
  codecFlagNoScreenUpdate       = (1L << 3), /* decompress*/
  codecFlagWasCompressed        = (1L << 4), /* compress*/
  codecFlagDontOffscreen        = (1L << 5), /* decompress*/
  codecFlagUpdatePreviousComp   = (1L << 6), /* compress*/
  codecFlagForceKeyFrame        = (1L << 7), /* compress*/
  codecFlagOnlyScreenUpdate     = (1L << 8), /* decompress*/
  codecFlagLiveGrab             = (1L << 9), /* compress*/
  codecFlagDiffFrame            = (1L << 9), /* decompress*/
  codecFlagDontUseNewImageBuffer = (1L << 10), /* decompress*/
  codecFlagInterlaceUpdate      = (1L << 11), /* decompress*/
  codecFlagCatchUpDiff          = (1L << 12), /* decompress*/
  codecFlagSupportDisable       = (1L << 13), /* decompress*/
  codecFlagReenable             = (1L << 14) /* decompress*/
};
```

Constants

`codecFlagUpdatePrevious`

Controls whether your compressor updates the previous image during compression. This flag is only used with sequences that are being temporally compressed. If this flag is set to 1, your compressor should copy the current frame into the previous frame buffer at the end of the frame-compression sequence. Use the source image.

Available in Mac OS X v10.0 and later.

Declared in `ImageCompression.h`.

`codecFlagWasCompressed`

Indicates to your compressor that the image to be compressed has been compressed before. This information may be useful to compressors that can compensate for the image degradation that may otherwise result from repeated compression and decompression of the same image. This flag is set to 1 to indicate that the image was previously compressed. This flag is set to 0 if the image was not previously compressed.

Available in Mac OS X v10.0 and later.

Declared in `ImageCompression.h`.

`codecFlagUpdatePreviousComp`

Controls whether your compressor updates the previous image buffer with the compressed image. This flag is only used with temporal compression. If this flag is set to 1, your compressor should update the previous frame buffer at the end of the frame-compression sequence, allowing your compressor to perform frame differencing against the compression results. Use the image that results from the compression operation. If this flag is set to 0, your compressor should not modify the previous frame buffer during compression.

Available in Mac OS X v10.0 and later.

Declared in `ImageCompression.h`.

`codecFlagLiveGrab`

Indicates whether the current sequence results from grabbing live video. When working with live video, your compressor should operate as quickly as possible and disable any additional processing, such as compensation for previously compressed data. This flag is set to 1 when you are compressing from a live video source.

Available in Mac OS X v10.0 and later.

Declared in `ImageCompression.h`.

`codecFlagDiffFrame`

Decompress.

Available in Mac OS X v10.0 and later.

Declared in `ImageCompression.h`.

`codecFlagSupportDisable`

Decompress.

Available in Mac OS X v10.0 and later.

Declared in `ImageCompression.h`.

Declared In

`ImageCompression.h`

Color Modes

Constants that represent color modes.

```
enum {
  defaultDither                 = 0,
  forceDither                   = 1,
  suppressDither                = 2,
  useColorMatching              = 4
};
enum {
  graphicsModeStraightAlpha     = 256,
  graphicsModePreWhiteAlpha     = 257,
  graphicsModePreBlackAlpha     = 258,
  graphicsModeComposition       = 259,
  graphicsModeStraightAlphaBlend = 260,
  graphicsModePreMulColorAlpha  = 261,
  graphicsModePerComponentAlpha = 272
};
enum {
  kQTTIFFUserDataPrefix         = 0x74690000, /* Added to some tag values in TIFF
 IFDs to generate user data codes.  (0x7469 is 'ti'.) */
                                        /* For example, YCbCrPositioning is tag
0x0213, so its user data code is 0x74690213. */
  kQTTIFFExifUserDataPrefix     = 0x65780000, /* Added to tag values in Exif IFDs
 to generate user data codes.  (0x6578 is 'ex'.) */
                                        /* For example, DateTimeOriginal is tag
0x9003, so its user data code is 0x65789003. */
  kQTTIFFExifGPSUserDataPrefix  = 0x67700000, /* Added to tag values in Exif GPS
IFDs to generate user data codes.  (0x6770 is 'gp'.) */
                                       /* For example, GPSAltitude is tag 0x0006,
 so its user data code is 0x6770006. */
  kQTAlphaMode                 = 'almo', /* UInt32; eg, graphicsModeStraightAlpha
 or graphicsModePreBlackAlpha */
  kQTAlphaModePreMulColor       = 'almp', /* RGBColor; used if kQTAlphaMode is
graphicsModePreMulColorAlpha */
  kUserDataIPTC                 = 'iptc'
};
```

Constants

`kQTTIFFUserDataPrefix`

Added to some tag values in TIFF IFDs to generate user data codes. (0x7469 is `'ti'`.).

Available in Mac OS X v10.1 and later.

Declared in `ImageCompression.h`.

`kQTTIFFExifUserDataPrefix`

Added to tag values in `Exif` IFDs to generate user data codes. (0x6578 is `'ex'`.).

Available in Mac OS X v10.1 and later.

Declared in `ImageCompression.h`.

`kQTTIFFExifGPSUserDataPrefix`

Added to tag values in `Exif` GPS IFDs to generate user data codes. (0x6770 is `'gp'`.).

Available in Mac OS X v10.1 and later.

Declared in `ImageCompression.h`.

`kQTAlphaMode`

UInt32; for example, `graphicsModeStraightAlpha` or `graphicsModePreBlackAlpha`.

Available in Mac OS X v10.1 and later.

Declared in `ImageCompression.h`.

`kQTAlphaModePreMulColor`
: `RGBColor`; used if `kQTAlphaMode` is `graphicsModePreMulColorAlpha`.

  Available in Mac OS X v10.1 and later.

  Declared in `ImageCompression.h`.

Declared In
`ImageCompression.h`

GetGraphicsImporterForFileWithFlags Values

Constants passed to GetGraphicsImporterForFileWithFlags.

```
enum {
  kDontUseValidateToFindGraphicsImporter = 1L << 0
};
```

Declared In
`ImageCompression.h`

QTSetPixMapPtrRequestedGammaLevel Values

Constants passed to QTSetPixMapPtrRequestedGammaLevel.

```
enum {
  kQTUsePlatformDefaultGammaLevel = 0,  /* When decompressing into this PixMap,
gamma-correct to the platform's standard gamma. */
  kQTUseSourceGammaLevel        = -1L,  /* When decompressing into this PixMap,
don't perform gamma-correction. */
  kQTCCIR601VideoGammaLevel     = 0x00023333 /* 2.2, standard television video
gamma.*/
};
```

Constants
`kQTCCIR601VideoGammaLevel`
: Gamma 2.2, for ITU-R BT.601 based video.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCompression.h`.

Declared In
`ImageCompression.h`

Data Components Reference for QuickTime

Framework:	Frameworks/QuickTime.framework
Declared in	QuickTimeComponents.h

Overview

Data components allow applications to place various types of data into a QuickTime movie or extract data from a movie in a specified format.

Functions by Task

Data Codec Functions

`DataCodecBeginInterruptSafe` (page 759)
: Called before performing a compression or decompression operation during interrupt time.

`DataCodecCompress` (page 760)
: Compresses data using the specified compressor component.

`DataCodecDecompress` (page 762)
: Decompresses data using the specified compressor component.

`DataCodecEndInterruptSafe` (page 764)
: Releases resources used by DataCodecBeginInterruptSafe.

Identifying Data References

`DataHCompareDataRef` (page 770)
: Compares a supplied data reference against its current data reference and returns a Boolean value indicating whether the data references are equivalent (that is, the two data references identify the same container).

`DataHGetDataRef` (page 780)
: Retrieves your component's current data reference.

`DataHResolveDataRef` (page 805)
: Locates the container associated with a given data reference.

`DataHSetDataRef` (page 809)
: Assigns a data reference to your data handler component.

Managing Data Handler Components

`DataHFlushCache` (page 775)
Discards the contents of any cached read buffers.

`DataHFlushData` (page 775)
Forces any data in your component's write buffers to be written to the device that contains the current data reference.

`DataHPlaybackHints` (page 798)
Provides additional information to your component that you may use to optimize the operation of your data handler.

`DataHTask` (page 816)
Cedes processor time to your data handler.

Reading Movie Data

`DataHCloseForRead` (page 768)
Closes read-only access to its data reference.

`DataHFinishData` (page 774)
Completes or cancels one or more queued read requests.

`DataHGetAvailableFileSize` (page 776)
Returns the available file size for a data handler component.

`DataHGetData` (page 777)
Reads data from its current data reference, which is a synchronous read operation.

`DataHGetFileSize` (page 784)
Returns the size, in bytes, of the current data reference.

`DataHGetScheduleAheadTime` (page 794)
Reports how far in advance it prefers clients to issue read requests.

`DataHOpenForRead` (page 796)
Opens a data handler's current data reference for read-only access.

`DataHScheduleData` (page 806)
Reads data from its current data reference, which can be a synchronous read operation or an asynchronous read operation.

Selecting a Data Handler

`DataHCanUseDataRef` (page 767)
Reports whether a data handler can access the data associated with a specified data reference.

`DataHGetDeviceIndex` (page 782)
Returns a value that identifies the device on which a data reference resides.

`DataHGetVolumeList` (page 795)
Returns a list of the volumes your component can access, along with flags indicating your component's capabilities for each volume.

Using Data References to Access Media

`DataHDeleteFile` (page 773)
: Deletes a data handler's data storage file.

`DataHGetInfo` (page 788)
: Retrieves information from a data handler.

`DataHSetMovieUsageFlags` (page 814)
: Sets the way that a data handler appends data to its storage.

Working With The Idle Manager

`DataHSetIdleManager` (page 813)
: Lets a data handler report its idling needs.

Writing Movie Data

`DataHCloseForWrite` (page 769)
: Closes write-only access to its data reference.

`DataHCreateFile` (page 771)
: Creates a new container that meets the specifications of the current data reference.

`DataHGetFreeSpace` (page 787)
: Reports the number of bytes available on the device that contains the current data reference.

`DataHGetPreferredBlockSize` (page 793)
: Reports the block size that it prefers to use when accessing the current data reference.

`DataHOpenForWrite` (page 797)
: Opens your component's current data reference for write-only access.

`DataHPreextend` (page 801)
: Allocates new space for the current data reference, enlarging the container.

`DataHPutData` (page 802)
: Writes data to a component's current data reference.

`DataHSetFileSize` (page 811)
: Sets the size, in bytes, of the current data reference.

`DataHWrite` (page 817)
: Writes data to its current data reference.

Supporting Functions

`DataCodecCompressPartial` (page 761)
: Undocumented

`DataCodecDecompressPartial` (page 763)
: Undocumented

`DataCodecGetCompressBufferSize` (page 765)
: Returns the maximum possible size of the compressed data that will be returned using the specified compressor component.

`DataHAddMovie` (page 766)
: Assigns movie data to a data handler.

`DataHAppend64` (page 766)
: Appends data to the current data reference.

`DataHCreateFileWithFlags` (page 772)
: Undocumented

`DataHDoesBuffer` (page 773)
: Reports whether a data handler does buffer reads and writes.

`DataHGetCacheSizeLimit` (page 776)
: Returns the cache size limit for a data handler component.

`DataHGetDataAvailability` (page 778)
: Undocumented

`DataHGetDataInBuffer` (page 779)
: Returns the information about the data in a data handler component's buffer.

`DataHGetDataRate` (page 779)
: Undocumented

`DataHGetDataRefAsType` (page 780)
: Retrieves a data handler component's current data reference of a given type.

`DataHGetDataRefExtension` (page 781)
: Retrieves your component's current data reference extension data.

`DataHGetDataRefWithAnchor` (page 782)
: Retrieves a data handler component's component's current data reference and anchor data reference.

`DataHGetFileName` (page 783)
: Retrieves the name of the file supplying the current data reference for a data handler.

`DataHGetFileSize64` (page 785)
: Provides a 64-bit version of DataHGetFileSize.

`DataHGetFileSizeAsync` (page 785)
: Returns the size of the current data reference, invoking a completion callback.

`DataHGetFileTypeOrdering` (page 786)
: Returns the preferred ordering for file typing information.

`DataHGetFreeSpace64` (page 787)
: Provides a 64-bit version of DataHGetFreeSpace.

`DataHGetInfoFlags` (page 789)
: Provides information about the operation of a data handler component.

`DataHGetMacOSFileType` (page 789)
: Gets the Mac OS file type for a data handler's current data reference.

`DataHGetMIMEType` (page 790)
: Gets the MIME type for a data handler's current data reference.

`DataHGetMIMETypeAsync` (page 790)
: Performs asynchronous discovery of a HTTP/FTP connection's MIME type.

`DataHGetMovie` (page 791)
: Gets the movie for a data handler's current data reference.

`DataHGetMovieWithFlags` (page 792)
: Gets the movie for a data handler's current data reference, allowing the flags that would be passed to NewMovieFromDataRef to be passed to the handler.

`DataHGetTemporaryDataRefCapabilities` (page 794)
: Undocumented

`DataHIsStreamingDataHandler` (page 796)
: Determines if a data handler handles streaming data.

`DataHPlaybackHints64` (page 799)
: Provides a 64-bit version of DataHPlaybackHints.

`DataHPollRead` (page 800)
: Undocumented

`DataHPreextend64` (page 801)
: Provides a 64-bit version of DataHPreextend.

`DataHReadAsync` (page 803)
: Undocumented

`DataHRenameFile` (page 804)
: Undocumented

`DataHScheduleData64` (page 807)
: Provides a 64-bit version of DataHScheduleData.

`DataHSetCacheSizeLimit` (page 808)
: Sets the cache size limit for a data handler component.

`DataHSetDataRefExtension` (page 810)
: Sets your component's current data reference extension data.

`DataHSetDataRefWithAnchor` (page 811)
: Sets the data reference and anchor data reference for a data handler.

`DataHSetFileSize64` (page 812)
: Provides a 64-bit version of DataHSetFileSize.

`DataHSetMacOSFileType` (page 813)
: Sets the Mac OS file type for a data handler's current data reference.

`DataHSetTimeBase` (page 814)
: Sets the time base for a data handler component.

`DataHSetTimeHints` (page 815)
: Undocumented

`DataHUpdateMovie` (page 816)
: Updates the movie for a data handler's current data reference.

`DataHUseTemporaryDataRef` (page 817)
: Undocumented

`DataHWrite64` (page 819)
: Provides a 64-bit version of DataHWrite.

`DisposeCDataHandlerUPP` (page 820)
: Disposes of a CDataHandlerUPP pointer.

`DisposeCharDataHandlerUPP` (page 820)
: Disposes of a CharDataHandlerUPP pointer.

`DisposeCommentHandlerUPP` (page 821)
: Disposes of a CommentHandlerUPP pointer.

`DisposeDataHCompletionUPP` (page 821)
: Disposes of a DataHCompletionUPP pointer.

`DisposeEndDocumentHandlerUPP` (page 822)
: Disposes of an EndDocumentHandlerUPP pointer.

`DisposeEndElementHandlerUPP` (page 822)
: Disposes of an EndElementHandlerUPP pointer.

`DisposePreprocessInstructionHandlerUPP` (page 822)
: Disposes of a PreprocessInstructionHandlerUPP pointer.

`DisposeStartDocumentHandlerUPP` (page 823)
: Disposes of a StartDocumentHandlerUPP pointer.

`DisposeStartElementHandlerUPP` (page 823)
: Disposes of a StartElementHandlerUPP pointer.

`DisposeVdigIntUPP` (page 824)
: Disposes of a VdigIntUPP pointer.

`NewCDataHandlerUPP` (page 824)
: Allocates a Universal Procedure Pointer for the CDataHandlerProc callback.

`NewCharDataHandlerUPP` (page 825)
: Undocumented

`NewCommentHandlerUPP` (page 825)
: Undocumented

`NewDataHCompletionUPP` (page 825)
: Allocates a Universal Procedure Pointer for the DataHCompletionProc callback.

`NewEndDocumentHandlerUPP` (page 826)
: Undocumented

`NewEndElementHandlerUPP` (page 827)
: Undocumented

`NewPreprocessInstructionHandlerUPP` (page 827)
: Undocumented

`NewStartDocumentHandlerUPP` (page 827)
: Undocumented

`NewStartElementHandlerUPP` (page 828)
: Undocumented

`NewVdigIntUPP` (page 828)
: Allocates a Universal Procedure Pointer for the VdigIntProc callback.

`XMLParseAddAttribute` (page 829)
: Undocumented

`XMLParseAddAttributeAndValue` (page 830)
: Undocumented

`XMLParseAddAttributeValueKind` (page 830)
: Undocumented

`XMLParseAddElement` (page 831)
  Undocumented

`XMLParseAddMultipleAttributes` (page 832)
  Undocumented

`XMLParseAddMultipleAttributesAndValues` (page 833)
  Undocumented

`XMLParseAddNameSpace` (page 833)
  Undocumented

`XMLParseDataRef` (page 834)
  Undocumented

`XMLParseDisposeXMLDoc` (page 835)
  Undocumented

`XMLParseFile` (page 835)
  Undocumented

`XMLParseGetDetailedParseError` (page 836)
  Undocumented

`XMLParseSetCDataHandler` (page 836)
  Undocumented

`XMLParseSetCharDataHandler` (page 837)
  Undocumented

`XMLParseSetCommentHandler` (page 837)
  Undocumented

`XMLParseSetEndDocumentHandler` (page 838)
  Undocumented

`XMLParseSetEndElementHandler` (page 838)
  Undocumented

`XMLParseSetEventParseRefCon` (page 839)
  Undocumented

`XMLParseSetOffsetAndLimit` (page 839)
  Undocumented

`XMLParseSetPreprocessInstructionHandler` (page 840)
  Undocumented

`XMLParseSetStartDocumentHandler` (page 841)
  Undocumented

`XMLParseSetStartElementHandler` (page 841)
  Undocumented

Functions

DataCodecBeginInterruptSafe

Called before performing a compression or decompression operation during interrupt time.

```
ComponentResult DataCodecBeginInterruptSafe (
   DataCodecComponent dc,
   unsigned long maxSrcSize
);
```

Parameters

dc

The instance of a compressor or decompressor component for this request. Your software obtains this reference when calling the Component Manager's `OpenComponent` or `OpenDefaultComponent` function.

maxSrcSize

The maximum size of a block of data to be compressed or decompressed, in bytes.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function allocates any temporary buffers that are necessary to perform the operation during interrupt time. To release the resources used to make the operation safe during interrupt time, call the `DataCodecEndInterruptSafe` (page 764) function or close the instance of the compressor or decompressor component.

Special Considerations

If the function returns an error, your software must not perform compression or decompression operations during interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

DataCodecCompress

Compresses data using the specified compressor component.

```
ComponentResult DataCodecCompress (
   DataCodecComponent dc,
   void *srcData,
   UInt32 srcSize,
   void *dstData,
   UInt32 dstBufferSize,
   UInt32 *actualDstSize,
   UInt32 *decompressSlop
);
```

Parameters

dc

The compressor component to used.

srcData

A pointer to the data to be compressed.

srcSize
: The size of the data to be compressed, in bytes.

dstData
: A pointer to the buffer in which to store the compressed data.

dstBufferSize
: The size of the buffer in which to store the compressed data, in bytes.

actualDstSize
: The size of the compressed data that was created, in bytes.

decompressSlop
: The number of bytes that should be added to the decompression buffer size if decompression occurs in place.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Before calling this function, you should call `DataCodecGetCompressBufferSize` (page 765) to obtain the maximum possible size of the compressed data that will be returned. You can then use this value as the value of the `dstBufferSize` parameter. Note that a buffer for compressed data that is the same size as the uncompressed data may not be large enough: in some cases, the size of the compressed data can be larger than the size of the decompressed data. When you compress data, you should store the size of data before compression at the beginning of the file, immediately before the compressed data. This allows you to obtain the size of the decompressed data and allocate the buffer for storing the decompressed data before calling `DataCodecDecompress` (page 762).

Special Considerations

You can compress sprites by calling this function. If you do, you must include the uncompressed size of the sample at the beginning of the sample, before the compressed data, and store the `component` subtype of the data compressor used to compress the sprite in the `decompressorType` field of the sample's `SpriteDescription` structure. You can also compress QuickDraw 3D media samples by calling this function.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qteffects.win

qtsprites.win

qtwiredactions

qtwiredsprites

qtwiredspritesjr

Declared In
`QuickTimeComponents.h`

DataCodecCompressPartial

Undocumented

```
ComponentResult DataCodecCompressPartial (
   DataCodecComponent dc,
   void **next_in,
   unsigned long *avail_in,
   unsigned long *total_in,
   void **next_out,
   unsigned long *avail_out,
   unsigned long *total_out,
   Boolean tryToFinish,
   Boolean *didFinish
);
```

Parameters

dc

The compressor component to used.

next_in

Undocumented

avail_in

Undocumented

total_in

Undocumented

next_out

Undocumented

avail_out

Undocumented

total_out

Undocumented

tryToFinish

Undocumented

didFinish

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

DataCodecDecompress

Decompresses data using the specified compressor component.

```
ComponentResult DataCodecDecompress (
   DataCodecComponent dc,
   void *srcData,
   UInt32 srcSize,
   void *dstData,
   UInt32 dstBufferSize
);
```

Parameters

dc

The decompressor component to used.

srcData

A pointer to the data to be decompressed.

srcSize

The size of the data to be decompressed, in bytes.

dstData

A pointer to the buffer in which to store the decompressed data.

dstBufferSize

The size of the buffer in which to store the decompressed data, in bytes.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Before allocating the buffer in which to store decompressed data, you need to get the size of the decompressed data. The size is normally stored at the beginning of the file, immediately before the compressed data.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

DataCodecDecompressPartial

Undocumented

```
ComponentResult DataCodecDecompressPartial (
   DataCodecComponent dc,
   void **next_in,
   unsigned long *avail_in,
   unsigned long *total_in,
   void **next_out,
   unsigned long *avail_out,
   unsigned long *total_out,
   Boolean *didFinish
);
```

Parameters

dc
: The decompressor component to used.

next_in
: *Undocumented*

avail_in
: *Undocumented*

total_in
: *Undocumented*

next_out
: *Undocumented*

avail_out
: *Undocumented*

total_out
: *Undocumented*

didFinish
: *Undocumented*

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

DataCodecEndInterruptSafe

Releases resources used by DataCodecBeginInterruptSafe.

```
ComponentResult DataCodecEndInterruptSafe (
   DataCodecComponent dc
);
```

Parameters

dc
: The instance of a compressor or decompressor component for this request.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
When your software has finished a compression or decompression operation that must be performed during interrupt time, it can call this function to release any memory of other resources that were used by `DataCodecBeginInterruptSafe`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

DataCodecGetCompressBufferSize

Returns the maximum possible size of the compressed data that will be returned using the specified compressor component.

```
ComponentResult DataCodecGetCompressBufferSize (
   DataCodecComponent dc,
   UInt32 srcSize,
   UInt32 *dstSize
);
```

Parameters

dc
The compressor component to used.

srcSize
The size in bytes of the data being compressed.

dstSize
A pointer to the maximum size of the compressed data that will be returned.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
The actual size of the compressed data will likely be smaller than the size you initially have to allocate, so after you compress the data you should shrink the compressed data handle down to the actual data size. When compressing a movie, allocate an extra ten 32-bit integers of space to store the compressed movie resource header information, as shown in the following code sample:

```
unsigned long compressedSize;
Handle compressedData;
DataCodecGetCompressBufferSize(dataCompressor, movieResourceSize,
&compressedSize);
compressedData =NewHandle(compressedSize + (10 * sizeof(UInt32)));
```

Version Notes
Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qteffects.win

qtsprites.win

qtwiredactions

qtwiredsprites

qtwiredspritesjr

Declared In

`QuickTimeComponents.h`

DataHAddMovie

Assigns movie data to a data handler.

```
ComponentResult DataHAddMovie (
   DataHandler dh,
   Movie theMovie,
   short *id
);
```

Parameters

dh

A data handler component.

theMovie

A movie identifier. Your application obtains this identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

id

A pointer to the field that specifies the resource containing the movie data that is to be added.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

DataHAppend64

Appends data to the current data reference.

```
ComponentResult DataHAppend64 (
   DataHandler dh,
   void *data,
   wide *fileOffset,
   unsigned long size
);
```

Parameters

dh

A data handler component.

data

A pointer to data to be appended.

fileOffset

A pointer to a 64-bit value that represents the offset in the file of data to be appended.

size

The size of the data to be appended.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

DataHCanUseDataRef

Reports whether a data handler can access the data associated with a specified data reference.

```
ComponentResult DataHCanUseDataRef (
   DataHandler dh,
   Handle dataRef,
   long *useFlags
);
```

Parameters

dh

Identifies the calling program's connection to your data handler component.

dataRef

The data reference. This parameter contains a handle to the information that identifies the container in question.

useFlags

A pointer to a field of flags (see below) that your data handler component uses to indicate its ability to access the container identified by the `dataRef` parameter. Set all appropriate flags to 1; set unused flags to 0. For example, if your component supports networked multimedia servers using a special set of protocols, your data handler should set the `kDataHCanRead` and `kDataHCanSpecialRead` flags to 1 for any container that is on that server. In addition, if your component can write to the server, set the `kDataHCanWrite` and `kDataHCanSpecialWrite` flags to 1 (perhaps along with `kDataHCanStreamingWrite`). If your data handler cannot access the container, set the whole field to 0. See these constants:

`kDataHCanRead`

`kDataHSpecialRead`

`kDataHSpecialReadFile`

`kDataHCanWrite`

`kDataHSpecialWrite`

`kDataHCanStreamingWrite`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Apple's standard data handler sets both the `kDataHCanRead` and `kDataHCanWrite` flags to 1 for any data reference it receives, indicating that it can read from and write to any volume.

Special Considerations

Your data handler may use any facilities necessary to determine whether it can access the container. Bear in mind, though, that your component should try to be as quick about this determination as possible, in order to minimize the chance that the delay will be noticed by the user.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

DataHCloseForRead

Closes read-only access to its data reference.

```
ComponentResult DataHCloseForRead (
   DataHandler dh
);
```

Parameters

dh

Identifies the calling program's connection to your data handler component.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Data handler components provide two basic read facilities: `DataHGetData` (page 777) function is a fully synchronous read operation, while `DataHScheduleData` (page 806) is asynchronous. Applications provide scheduling information when they call your component's `DataHScheduleData` function. When your component processes the queued request, it calls the application's data-handler completion function. Before any application can read data from a data reference, it must open read access to that reference by calling your component's `DataHOpenForRead` (page 796) function. `DataHCloseForRead` (page 768) closes that read access path.

Special Considerations
Note that a client program may close its connection to your component (by calling `CloseComponent`) without closing the read path. If this happens, your component should close the data reference before closing the connection.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qtdataref

qtfiletransfer

qtfiletransfer.win

ThreadsImporter

ThreadsImportMovie

Declared In
`QuickTimeComponents.h`

DataHCloseForWrite

Closes write-only access to its data reference.

```
ComponentResult DataHCloseForWrite (
   DataHandler dh
);
```

Parameters
dh

Identifies the calling program's connection to your data handler component.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Data handlers provide two distinct write facilities: `DataHPutData` (page 802) is a simple synchronous interface that allows applications to append data to the end of a container, and `DataHWrite` is a more capable, asynchronous write function that is suitable for movie capture operations. Before writing data to a data reference, applications must call your component's `DataHOpenForWrite` (page 797) function to open a write path to the container. `DataHCloseForWrite` closes that write path. As is the case with `DataHScheduleData` (page 806), your component calls the application's data-handler completion function when you are done with the write request.

Special Considerations

A client program may close its connection to your component (by calling `CloseComponent`) without closing the write path. If this happens, your component should close the data reference before closing the connection. Note that some data handlers may not support write operations. For example, some shared devices, such as a CD-ROM "jukebox," may be read-only devices. As a result, it is very important that your data handler correctly report its write capabilities to client programs.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qtdataref

qtfiletransfer

qtfiletransfer.win

ThreadsImporter

ThreadsImportMovie

Declared In
`QuickTimeComponents.h`

DataHCompareDataRef

Compares a supplied data reference against its current data reference and returns a Boolean value indicating whether the data references are equivalent (that is, the two data references identify the same container).

```
ComponentResult DataHCompareDataRef (
   DataHandler dh,
   Handle dataRef,
   Boolean *equal
);
```

Parameters

dh

Identifies the calling program's connection to your data handler component.

dataRef

The data reference to be compared to your component's current data reference. Different types of containers may require different types of data references. For example, a reference to a memory-based movie may be a handle, while a reference to a file-based movie may be an alias. Apple's memory-based data handler for the Macintosh uses handles (and has a subtype value of `'hndl'`), while the HFS data handler uses Alias Manager aliases (its subtype value is `'alis'`). See `Data References`.

equal

A pointer to a Boolean. Your component should set that Boolean to TRUE if the two data references identify the same container. Otherwise, set the Boolean to FALSE.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion

All data handler components use data references to identify and locate a movie's container. `DataHCompareDataRef` asks your component to compare a data reference against the current data reference and indicate whether the references are equivalent (that is, refer to the same container). Client programs can correlate data references with data handlers by matching the component's subtype value with the `data` reference type; the subtype value indicates the type of data reference the component supports. All data handlers with the same subtype value must support the same data reference type.

Special Considerations

Note that your component cannot simply compare the bits in the two data references. For example, two completely different aliases may refer to the same HFS file. Consequently, you need to completely resolve the data reference in order to determine the file identified by the reference.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

DataHCreateFile

Creates a new container that meets the specifications of the current data reference.

```
ComponentResult DataHCreateFile (
   DataHandler dh,
   OSType creator,
   Boolean deleteExisting
);
```

Parameters

dh

Identifies the calling program's connection to your data handler component.

creator

The creator type of the new container. If the client program sets this parameter to 0, your component should choose a reasonable value (for example, 'TV0D', the `creator` type for Apple's movie player).

deleteExisting

Indicates whether to delete any existing data. If this parameter is set to TRUE and a container already exists for the current data reference, your component should delete that data before creating the new container. If this parameter is set to FALSE, your component should preserve any data that resides in the container defined by the current data reference (if there is any).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Data handlers provide two distinct write facilities: `DataHPutData` (page 802) is a simple synchronous interface that allows applications to append data to the end of a container, while `DataHWrite` is a more capable, asynchronous write function that is suitable for movie capture operations. As is the case with `DataHScheduleData` (page 806), your component calls the application's data-handler completion function when you are done with the write request.

Special Considerations

Note that some data handlers may not support write operations. For example, some shared devices, such as a CD-ROM "jukebox," may be read-only devices. As a result, it is very important that your data handler correctly report its write capabilities to client programs.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

ElectricImageComponent

ElectricImageComponent.win

Declared In

`QuickTimeComponents.h`

DataHCreateFileWithFlags

Undocumented

```
ComponentResult DataHCreateFileWithFlags (
   DataHandler dh,
   OSType creator,
   Boolean deleteExisting,
   UInt32 flags
);
```

Parameters

dh

Identifies the calling program's connection to your data handler component.

creator

The creator type of the new container. If the client program sets this parameter to 0, your component should choose a reasonable value (for example, 'TV0D', the `creator` type for Apple's movie player).

deleteExisting

Indicates whether to delete any existing data. If this parameter is set to TRUE and a container already exists for the current data reference, your component should delete that data before creating the new container. If this parameter is set to FALSE, your component should preserve any data that resides in the container defined by the current data reference (if there is any).

flags

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

DataHDeleteFile

Deletes a data handler's data storage file.

```
ComponentResult DataHDeleteFile (
   DataHandler dh
);
```

Parameters

dh
A data handler component.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
QTExtractAndConvertToMovieFile

Declared In
`QuickTimeComponents.h`

DataHDoesBuffer

Reports whether a data handler does buffer reads and writes.

```
ComponentResult DataHDoesBuffer (
   DataHandler dh,
   Boolean *buffersReads,
   Boolean *buffersWrites
);
```

Parameters

dh
A data handler component.

buffersReads
A pointer to a Boolean that is returned true if the data handler component does buffer reads, false otherwise.

buffersWrites
A pointer to a Boolean that is returned true if the data handler component does buffer writes, false otherwise.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

DataHFinishData

Completes or cancels one or more queued read requests.

```
ComponentResult DataHFinishData (
   DataHandler dh,
   Ptr PlaceToPutDataPtr,
   Boolean Cancel
);
```

Parameters

`dh`

Identifies the calling program's connection to your data handler component.

`PlaceToPutDataPtr`

The location in memory that is to receive the data. The value of this parameter identifies the specific read request to be completed. If this parameter is set to `NIL`, the call affects all pending read requests.

`Cancel`

Indicates whether the calling program wants to cancel the outstanding request. If this parameter is set to TRUE, your data handler should cancel the request (or requests) identified by the `PlaceToPutDataPtr` parameter.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Client programs use `DataHFinishData` either to cancel outstanding read requests or to demand that the requests be serviced immediately. Note that your component must call the client program's data-handler completion function for each queued request, even though the client program called `DataHFinishData`. Be sure to call the completion function for both canceled and completed read requests.

Special Considerations

Preroll operations are a special case of the immediate service request. The client program will have queued one or more read requests with their scheduled time of delivery set infinitely far into the future. Your data handler queues those requests until the client program calls `DataHFinishData` demanding that all outstanding read requests be satisfied immediately.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

DataHFlushCache

Discards the contents of any cached read buffers.

```
ComponentResult DataHFlushCache (
   DataHandler dh
);
```

Parameters

dh

Identifies the calling program's connection to your data handler component.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Note that this function does not invalidate any queued read requests (made by calling your component's `DataHScheduleData` (page 806) function).

Special Considerations

Client programs may call this function if they have, in some way, changed the container associated with the current data reference on their own. Under these circumstances, data your component may have read and cached in anticipation of future read requests from the client may be invalid.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

DataHFlushData

Forces any data in your component's write buffers to be written to the device that contains the current data reference.

```
ComponentResult DataHFlushData (
   DataHandler dh
);
```

Parameters

dh

Identifies the calling program's connection to your data handler component.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function is essentially analogous to the Mac OS File Manager's `PBFlushFile` function. The client program may call this function after any write operation (either `DataHPutData` (page 802) or `DataHWrite` (page 817)). Your component should do what is necessary to make sure that the data is written to the storage device that contains the current data reference.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

DataHGetAvailableFileSize

Returns the available file size for a data handler component.

```
ComponentResult DataHGetAvailableFileSize (
   DataHandler dh,
   long *fileSize
);
```

Parameters

dh
: A data handler component.

fileSize
: A pointer to the file size.

Return Value
See `Error Codes`. Returns `badComponentSelector` if the data handler component does not support this call. Returns `noErr` if there is no error.

Discussion
You can call this function during an asynchronous read operation, after calling `DataHScheduleData` (page 806).

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

DataHGetCacheSizeLimit

Returns the cache size limit for a data handler component.

```
ComponentResult DataHGetCacheSizeLimit (
   DataHandler dh,
   Size *cacheSizeLimit
);
```

Parameters

dh
: A data handler component.

cacheSizeLimit

A pointer to the cache size limit.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

DataHGetData

Reads data from its current data reference, which is a synchronous read operation.

```
ComponentResult DataHGetData (
   DataHandler dh,
   Handle h,
   long hOffset,
   long offset,
   long size
);
```

Parameters

dh

Identifies the calling program's connection to your data handler component.

h

The handle to receive the data.

hOffset

Identifies the offset into the handle where your component should return the data.

offset

The offset in the data reference from which your component is to read.

size

The number of bytes to read.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Data handler components provide two basic read facilities. `DataHGetData` function is a fully synchronous read operation, while `DataHScheduleData` (page 806) is asynchronous. (Applications provide scheduling information when they call `DataHScheduleData`.) Before any application can read data from a data reference, it must open read access to that reference by calling your component's `DataHOpenForRead` (page 796) function. `DataHCloseForRead` (page 768) closes that read access path. When your component processes the queued request, it calls the application's data-handler completion function.

Special Considerations

Note that the Movie Toolbox may try to read data from a data reference without calling your component's `DataHOpenForRead` (page 796) function. If this happens, your component should open the data reference for read-only access, respond to the read request, and then leave the data reference open in anticipation of later read requests.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

DataHGetDataAvailability

Undocumented

```
ComponentResult DataHGetDataAvailability (
   DataHandler dh,
   long offset,
   long len,
   long *missing_offset,
   long *missing_len
);
```

Parameters

`dh`

A data handler component.

`offset`

Undocumented

`len`

Undocumented

`missing_offset`

Undocumented

`missing_len`

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

DataHGetDataInBuffer

Returns the information about the data in a data handler component's buffer.

```
ComponentResult DataHGetDataInBuffer (
   DataHandler dh,
   long startOffset,
   long *size
);
```

Parameters

dh

A data handler component.

startOffset

The offset to the start of the data.

size

The size of the data.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

DataHGetDataRate

Undocumented

```
ComponentResult DataHGetDataRate (
   DataHandler dh,
   long flags,
   long *bytesPerSecond
);
```

Parameters

dh

A data handler component.

flags

Undocumented

bytesPerSecond

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

DataHGetDataRef

Retrieves your component's current data reference.

```
ComponentResult DataHGetDataRef (
   DataHandler dh,
   Handle *dataRef
);
```

Parameters

`dh`

Identifies the calling program's connection to your data handler component.

`dataRef`

A pointer to a data reference handle. Your component should make a copy of its current data reference in a handle and return that handle in this field. The client program is responsible for disposing of that handle. Different types of containers may require different types of data references. For example, a reference to a memory-based movie may be a handle, while a reference to a file-based movie may be an alias. Apple's memory-based data handler for the Macintosh uses handles (and has a subtype value of `'hndl'`), while the HFS data handler uses Alias Manager aliases (its subtype value is `'alis'`). See `Data References`.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
All data handler components use data references to identify and locate a movie's container. Client programs can correlate data references with data handlers by matching the component's subtype value with the `data` reference type; the subtype value indicates the type of data reference the component supports. All data handlers with the same subtype value must support the same data reference type.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

DataHGetDataRefAsType

Retrieves a data handler component's current data reference of a given type.

```
ComponentResult DataHGetDataRefAsType (
   DataHandler dh,
   OSType requestedType,
   Handle *dataRef
);
```

Parameters

dh

A data handler component.

requestedType

The type of the data reference to retrieve; see `Data References`.

dataRef

A pointer to a data reference handle. Your component should make a copy of its current data reference in a handle and return that handle in this field. The client program is responsible for disposing of that handle. Different types of containers may require different types of data references. For example, a reference to a memory-based movie may be a handle, while a reference to a file-based movie may be an alias. Apple's memory-based data handler for the Macintosh uses handles (and has a subtype value of `'hndl'`), while the HFS data handler uses Alias Manager aliases (its subtype value is `'alis'`). See `Data References`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.1.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

DataHGetDataRefExtension

Retrieves your component's current data reference extension data.

```
ComponentResult DataHGetDataRefExtension (
   DataHandler dh,
   Handle *extension,
   OSType idType
);
```

Parameters

dh

A data handler component.

extension

A pointer to a handle to the extension data.

idType

A four-byte signature identifying the type of extension data.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

DataHGetDataRefWithAnchor

Retrieves a data handler component's component's current data reference and anchor data reference.

```
ComponentResult DataHGetDataRefWithAnchor (
   DataHandler dh,
   Handle anchorDataRef,
   OSType dataRefType,
   Handle *dataRef
);
```

Parameters

dh

A data handler component.

anchorDataRef

A handle to the anchor data reference.

dataRefType

The type of the data reference; see `Data References`.

dataRef

A pointer to a data reference handle. Your component should make a copy of its current data reference in a handle and return that handle in this field. The client program is responsible for disposing of that handle. Different types of containers may require different types of data references. For example, a reference to a memory-based movie may be a handle, while a reference to a file-based movie may be an alias. Apple's memory-based data handler for the Macintosh uses handles (and has a subtype value of `'hndl'`), while the HFS data handler uses Alias Manager aliases (its subtype value is `'alis'`). See `Data References`.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

DataHGetDeviceIndex

Returns a value that identifies the device on which a data reference resides.

```
ComponentResult DataHGetDeviceIndex (
   DataHandler dh,
   long *deviceIndex
);
```

Parameters

dh

Identifies the calling program's connection to your data handler component.

deviceIndex

A pointer to a field that your data handler component uses to return a device identifier value. Your component may use any identifier value that is appropriate (for example, Apple's HFS data handler uses the volume reference number). The client program should do nothing with this value other than compare it with other identifiers returned by your data handler component.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Some client programs may need to account for the fact that two or more data references reside on the same device. For instance, this may affect storage-allocation requirements. This function allows such client programs to obtain this information from your data handler.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

DataHGetFileName

Retrieves the name of the file supplying the current data reference for a data handler.

```
ComponentResult DataHGetFileName (
   DataHandler dh,
   Str255 str
);
```

Parameters

dh

A data handler component.

str

The name of the file as a string.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

DataHGetFileSize

Returns the size, in bytes, of the current data reference.

```
ComponentResult DataHGetFileSize (
   DataHandler dh,
   long *fileSize
);
```

Parameters

dh

Identifies the calling program's connection to your data handler component.

fileSize

A pointer to a long integer. Your component returns the size of the container corresponding to the current data reference, in bytes.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function is operationally equivalent to the Mac OS File Manager's `GetEOF` function. Before writing data to a data reference, applications must call your component's `DataHOpenForWrite` (page 797) function to open a write path to the container. `DataHCloseForWrite` (page 769) closes that write path. As is the case with `DataHScheduleData` (page 806), your component calls the application's data-handler completion function when you are done with the write request.

Special Considerations

Note that some data handlers may not support write operations. For example, some shared devices, such as a CD-ROM "jukebox," may be read-only devices. As a result, it is very important that your data handler correctly report its write capabilities to client programs.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

ElectricImageComponent.win

qtdataref

qtfiletransfer

qtfiletransfer.win

ThreadsImportMovie

Declared In

`QuickTimeComponents.h`

DataHGetFileSize64

Provides a 64-bit version of DataHGetFileSize.

```
ComponentResult DataHGetFileSize64 (
   DataHandler dh,
   wide *fileSize
);
```

Parameters

dh

Identifies the calling program's connection to your data handler component.

fileSize

A pointer to a wide. Your component returns the size of the container corresponding to the current data reference, in bytes.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The only difference between this function and `DataHGetFileSize` (page 784) is that the `fileSize` parameter is a 64-bit integer instead of a 32-bit integer.

Special Considerations

New applications should use this function instead of the 32-bit version.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

DataHGetFileSizeAsync

Returns the size of the current data reference, invoking a completion callback.

```
ComponentResult DataHGetFileSizeAsync (
   DataHandler dh,
   wide *fileSize,
   DataHCompletionUPP completionRtn,
   long refCon
);
```

Parameters

dh

Identifies the calling program's connection to your data handler component.

fileSize

A pointer to a 64-bit value. Your component returns the size of the container corresponding to the current data reference, in bytes.

completionRtn

A pointer to a data handler completion callback, described in `DataHCompletionProc`.

refCon

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

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

DataHGetFileTypeOrdering

Returns the preferred ordering for file typing information.

```
ComponentResult DataHGetFileTypeOrdering (
   DataHandler dh,
   DataHFileTypeOrderingHandle *orderingListHandle
);
```

Parameters

dh

Identifies the calling program's connection to your data handler component.

orderingListHandle

A pointer to a handle to a list of `OSType` values (see below). The list may contain only a subset of the currently defined types (i.e., Mac OS file type, extension, MIME type) to limit the consideration to reasonable types. See these constants:

`kDataHFileTypeMacOSFileType`

`kDataHFileTypeExtension`

`kDataHFileTypeMIME`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

If the data handler has not set the data reference, it can choose to return either an error or a reasonable default ordering list.

Special Considerations

Before making a call to this function, the client should have opened the data handler and called `DataHSetDataRef` (page 809) or `DataHSetDataRefWithAnchor` (page 811). This allows the data handler to return a different ordering based on the particular file. This might allow for a data handler to vary its ordering based on the location of the file. For example, on the Mac OS, it might use extensions only on foreign volumes. For other volumes, it might use a Mac OS file type followed by a file extension.

Version Notes

Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

DataHGetFreeSpace

Reports the number of bytes available on the device that contains the current data reference.

```
ComponentResult DataHGetFreeSpace (
   DataHandler dh,
   unsigned long *freeSize
);
```

Parameters

dh

Identifies the calling program's connection to your data handler component.

freeSize

A pointer to an unsigned long integer. Your component returns the number of bytes of free space available on the device that contains the container referred to by the current data reference.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Before writing data to a data reference, applications must call your component's `DataHOpenForWrite` (page 797) function to open a write path to the container. `DataHCloseForWrite` (page 769) closes that write path. As is the case with `DataHScheduleData` (page 806), your component calls the application's data-handler completion function when you are done with the write request.

Special Considerations

Note that some data handlers may not support write operations. For example, some shared devices, such as a CD-ROM "jukebox," may be read-only devices. As a result, it is very important that your data handler correctly report its write capabilities to client programs.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

DataHGetFreeSpace64

Provides a 64-bit version of DataHGetFreeSpace.

```
ComponentResult DataHGetFreeSpace64 (
   DataHandler dh,
   wide *freeSize
);
```

Parameters

dh

A data handler component.

freeSize

A pointer to a 64-bit integer. Your component returns the number of bytes of free space available on the device that contains the container referred to by the current data reference.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The only difference between this function and `DataHGetFreeSpace` (page 787) is that the `freeSize` parameter is a 64-bit integer instead of a 32-bit integer.

Special Considerations

New applications should use this function instead of the 32-bit version.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

DataHGetInfo

Retrieves information from a data handler.

```
ComponentResult DataHGetInfo (
   DataHandler dh,
   OSType what,
   void *info
);
```

Parameters

dh

A data handler component.

what

A selector for the information requested. No selectors are currently defined.

info

A pointer to an area of memory in which to place the retrieved information.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.1 and later.

Declared In
`QuickTimeComponents.h`

DataHGetInfoFlags

Provides information about the operation of a data handler component.

```
ComponentResult DataHGetInfoFlags (
   DataHandler dh,
   UInt32 *flags
);
```

Parameters

dh
A data handler component.

flags
Flags (see below) that provide information about the data handler. See these constants:
- `kDataHInfoFlagNeverStreams`
- `kDataHInfoFlagCanUpdateDataRefs`
- `kDataHInfoFlagNeedsNetworkBandwidth`

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Fiendishthngs

Declared In
`QuickTimeComponents.h`

DataHGetMacOSFileType

Gets the Mac OS file type for a data handler's current data reference.

```
ComponentResult DataHGetMacOSFileType (
   DataHandler dh,
   OSType *fileType
);
```

Parameters

dh
A data handler component.

fileType

A pointer to a file type; see `File Types and Creators`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

ThreadsImportMovie

Declared In

`QuickTimeComponents.h`

DataHGetMIMEType

Gets the MIME type for a data handler's current data reference.

```
ComponentResult DataHGetMIMEType (
   DataHandler dh,
   Str255 mimeType
);
```

Parameters

dh

A data handler component.

mimeType

A MIME type string.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

ThreadsImportMovie

Declared In

`QuickTimeComponents.h`

DataHGetMIMETypeAsync

Performs asynchronous discovery of a HTTP/FTP connection's MIME type.

```
ComponentResult DataHGetMIMETypeAsync (
   DataHandler dh,
   Str255 mimeType,
   DataHCompletionUPP completionRtn,
   long refCon
);
```

Parameters

dh

A data handler component.

mimeType

The MIME type string when (and if) it becomes available.

completionRtn

A `DataHCompletionProc` callback that is called when either the data becomes available or there is a failure. Failure can happen if there is a timeout or `DataHFinishData` (page 774) is called with a cancel instruction. The `mimeType` parameter will not be updated until the complete routine executes. If a completion routine is not specified, the call will return immediately.

refCon

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

Return Value

If the MIME type is known, this function updates `mimeType` and returns `noErr`. If the information is not known yet, the error `notEnoughDataErr` is returned. This allows non-blocking calls to be made to this function. If it returns another error, that indicates some other failure; see `Error Codes`.

Discussion

This function removes synchronous blocks from QuickTime's movie opening code.

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

DataHGetMovie

Gets the movie for a data handler's current data reference.

```
ComponentResult DataHGetMovie (
   DataHandler dh,
   Movie *theMovie,
   short *id
);
```

Parameters

dh

A data handler component.

theMovie

A movie identifier. This is the same as the identifier your application obtains from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

id

A pointer to the field that specifies the resource containing the movie data.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

DataHGetMovieWithFlags

Gets the movie for a data handler's current data reference, allowing the flags that would be passed to NewMovieFromDataRef to be passed to the handler.

```
ComponentResult DataHGetMovieWithFlags (
   DataHandler dh,
   Movie *theMovie,
   short *id,
   short flags
);
```

Parameters

dh

A data handler component.

theMovie

A movie identifier. This is the same as the identifier your application obtains from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

id

A pointer to the field that specifies the resource containing the movie data.

flags

Flags (see below) that control movie characteristics. See these constants:

`newMovieActive`

`newMovieDontResolveDataRefs`

`newMovieDontAskUnresolvedDataRefs`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Passing these flags lets you control whether or not the movie returned is active. Additionally, it allows `async` movie loading to be more efficient.

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

DataHGetPreferredBlockSize

Reports the block size that it prefers to use when accessing the current data reference.

```
ComponentResult DataHGetPreferredBlockSize (
   DataHandler dh,
   long *blockSize
);
```

Parameters

dh

Identifies the calling program's connection to your data handler component.

blockSize

A pointer to a long integer. Your component returns the size of blocks (in bytes) it prefers to use when accessing the current data reference.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Different devices use different file system block sizes. This function allows your component to report its preferred block size to the client program. Note that the client program is not required to use this block size when making requests. Some clients may, however, try to accommodate your component's preference. Applications may call your component's `DataHGetPreferredBlockSize` function in order to determine how best to interact with your data handler. As is the case with `DataHScheduleData` (page 806), your component calls the application's data-handler completion function when you are done with the write request.

Special Considerations

Note that some data handlers may not support write operations. For example, some shared devices, such as a CD-ROM "jukebox," may be read-only devices. As a result, it is very important that your data handler correctly report its write capabilities to client programs.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

DataHGetScheduleAheadTime

Reports how far in advance it prefers clients to issue read requests.

```
ComponentResult DataHGetScheduleAheadTime (
   DataHandler dh,
   long *millisecs
);
```

Parameters

dh

Identifies the calling program's connection to your data handler component.

millisecs

A pointer to a long integer. Your component should set this field with a value indicating the number of milliseconds you prefer to receive read requests in advance of the time when the data must be delivered.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function allows your data handler to tell the client program how far in advance it should schedule its read requests. By default, the Movie Toolbox issues scheduled read requests between 1 and 2 seconds before it needs the data from those requests. For some data handlers, however, this may not be enough time. For example, some data handlers may have to accommodate network delays when processing read requests. Client programs that call `DataHGetScheduleAheadTime` may try to respect your component's preference.

Special Considerations

Note that not all client programs will call `DataHGetScheduleAheadTime`. Further, some clients may not be able to accommodate your preferred time in all cases, even if they have asked for your component's preference. As a result, your component should have a strategy for handling requests that don't provide enough advanced scheduling time. For example, if your component receives a `DataHScheduleData` (page 806) request that it cannot satisfy, it can fail the request with an appropriate error code.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

DataHGetTemporaryDataRefCapabilities

Undocumented

```
ComponentResult DataHGetTemporaryDataRefCapabilities (
   DataHandler dh,
   long *outUnderstoodFlags
);
```

Parameters

dh

A data handler component.

outUnderstoodFlags

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 6.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`QuickTimeComponents.h`

DataHGetVolumeList

Returns a list of the volumes your component can access, along with flags indicating your component's capabilities for each volume.

```
ComponentResult DataHGetVolumeList (
   DataHandler dh,
   DataHVolumeList *volumeList
);
```

Parameters

dh

Identifies the calling program's connection to your data handler component.

volumeList

A pointer to a field that your data handler component uses to return a handle to a volume list. Your component constructs the volume list by allocating a handle and filling it with a series of `DataHVolumeListRecord` structures, one structure for each volume your component can access.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

To reduce the delay that may result from choosing an appropriate data handler for a volume, the Movie Toolbox maintains a list of data handlers and the volumes they support. The Movie Toolbox uses `DataHGetVolumeList` to build that list. Your data handler may use any facilities necessary to determine whether it can access the volume, including opening a container on the volume. Your component should set to 1 as many of the capability flags as are appropriate for each volume. Don't include records for volumes your handler cannot support. For example, if your component supports networked multimedia servers using a special set of protocols, your data handler should set the `kDataHCanRead` and `kDataHCanSpecialRead` flags to 1 for any volume that is on that server. In addition, if your component can write to a volume on the server, set the `kDataHCanWrite` and `kDataHCanSpecialWrite` flags to 1 (perhaps along with `kDataHCanStreamingWrite`). However, your component should create entries only for those volumes that support your protocols.

Special Considerations

It is the calling program's responsibility to dispose of the handle returned by your component. The Movie Toolbox tracks mounting and unmounting removable volumes, and keeps its volume list current. As a result, the Movie Toolbox may call your component's `DataHGetVolumeList` function whenever a removable volume is mounted. If your data handler does not process data that is stored in file system volumes, you need not support this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

DataHIsStreamingDataHandler

Determines if a data handler handles streaming data.

```
ComponentResult DataHIsStreamingDataHandler (
   DataHandler dh,
   Boolean *yes
);
```

Parameters

dh

A data handler component.

yes

Pointer to a Boolean that returns TRUE if the data handler handles streaming data, FALSE otherwise.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

DataHOpenForRead

Opens a data handler's current data reference for read-only access.

```
ComponentResult DataHOpenForRead (
   DataHandler dh
);
```

Parameters

dh

Identifies the calling program's connection to your data handler component.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

After setting your component's current data reference by calling `DataHSetDataRef` (page 809), client programs call `DataHOpenForRead` to start reading from the data reference. Your component should open the data reference for read-only access. If the data reference is already open or cannot be opened, return an appropriate error code. As is the case with `DataHScheduleData` (page 806), your component calls the application's data-handler completion function when you are done with the write request.

Special Considerations

Note that the Movie Toolbox may try to read data from a data reference without calling your component's `DataHOpenForRead` function. If this happens, your component should open the data reference for read-only access, respond to the read request, and then leave the data reference open in anticipation of later read requests.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

ElectricImageComponent.win

qtdataref

qtfiletransfer

qtfiletransfer.win

ThreadsImportMovie

Declared In

`QuickTimeComponents.h`

DataHOpenForWrite

Opens your component's current data reference for write-only access.

```
ComponentResult DataHOpenForWrite (
   DataHandler dh
);
```

Parameters

dh

Identifies the calling program's connection to your data handler component.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion
After setting your component's current data reference by calling `DataHSetDataRef` (page 809), client programs call `DataHOpenForWrite` to start writing to the data reference. Your component should open the data reference for write-only access. If the data reference is already open or cannot be opened, return an appropriate error code. As is the case with `DataHScheduleData` (page 806), your component calls the application's data-handler completion function when you are done with the write request.

Special Considerations

Note that some data handlers may not support write operations. For example, some shared devices, such as a CD-ROM "jukebox," may be read-only devices. As a result, it is very important that your data handler correctly report its write capabilities to client programs.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
ElectricImageComponent.win
qtdataref
qtfiletransfer
qtfiletransfer.win
ThreadsImportMovie

Declared In
`QuickTimeComponents.h`

DataHPlaybackHints

Provides additional information to your component that you may use to optimize the operation of your data handler.

```
ComponentResult DataHPlaybackHints (
   DataHandler dh,
   long flags,
   unsigned long minFileOffset,
   unsigned long maxFileOffset,
   long bytesPerSecond
);
```

Parameters

dh
Identifies the calling program's connection to your data handler component.

flags
Reserved. Set this parameter to 0.

minFileOffset
Together with the `maxFileOffset` parameter, specifies the range of data the client program anticipates using from the current data reference. This parameter specifies the offset of earliest byte the program expects to use (that is, the minimum container offset value). If the client expects to access bytes from the beginning of the container, it should set this parameter to 0.

maxFileOffset

The offset of the latest byte the program expects to use (that is, the maximum container offset value). If the client expects to use bytes throughout the container, the client should set this parameter to -1.

bytesPerSecond

The rate in bytes per second at which your data handler must read data from the data reference in order to keep up with the client program's anticipated needs.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Applications may call your handler's `DataHPlaybackHints` function to provide you with some guidelines about how those applications plan to use the current data reference. Your component should be prepared to have this function called more than once for a given data reference. For example, the Movie Toolbox calls this function whenever a movie's playback rate changes. This is a handy way for your data handler to track playback rate changes.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

DataHPlaybackHints64

Provides a 64-bit version of DataHPlaybackHints.

```
ComponentResult DataHPlaybackHints64 (
   DataHandler dh,
   long flags,
   const wide *minFileOffset,
   const wide *maxFileOffset,
   long bytesPerSecond
);
```

Parameters

dh

Identifies the calling program's connection to your data handler component.

flags

Reserved. Set this parameter to 0.

minFileOffset

Together with the `maxFileOffset` parameter, specifies the range of data the client program anticipates using from the current data reference. This parameter points to the offset of the earliest byte the program expects to use (that is, the minimum container offset value). If the client expects to access bytes from the beginning of the container, it should set this parameter to 0.

maxFileOffset

Pointer to the offset of the latest byte the program expects to use (that is, the maximum container offset value). If the client expects to use bytes throughout the container, the client should set this parameter to -1.

bytesPerSecond

The rate in bytes per second at which your data handler must read data from the data reference in order to keep up with the client program's anticipated needs.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The only difference between this function and `DataHPlaybackHints` (page 798) is that the `minFileOffset` parameter and `maxFileOffset` parameter are 64-bit integers instead of 32-bit integers.

Special Considerations

New applications should use this function instead of the 32-bit version.

Version Notes

Introduced in QuickTime 4.1.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

DataHPollRead

Undocumented

```
ComponentResult DataHPollRead (
   DataHandler dh,
   void *dataPtr,
   UInt32 *dataSizeSoFar
);
```

Parameters

dh

A data handler component.

dataPtr

Undocumented

dataSizeSoFar

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

DataHPreextend

Allocates new space for the current data reference, enlarging the container.

```
ComponentResult DataHPreextend (
   DataHandler dh,
   unsigned long maxToAdd,
   unsigned long *spaceAdded
);
```

Parameters

dh

Identifies the calling program's connection to your data handler component.

maxToAdd

The amount of space to add to the current data reference, in bytes. If the client program sets this parameter to 0, your component should add as much space as it can.

spaceAdded

A pointer to an unsigned long integer. Your component returns the number of bytes it was able to add to the data reference, in bytes.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function asks your component to make a container larger, analogous to the Mac OS File Manager's `PBAllocContig` function. When it is called, your component should allocate contiguous free space. If there is not sufficient contiguous free space to satisfy the request, your component should return a `dskFulErr` error code. Client programs use this function in order to avoid incurring any space-allocation delay when capturing movie data. As is the case with `DataHScheduleData` (page 806), your component calls the application's data-handler completion function when you are done with the write request.

Special Considerations

Note that some data handlers may not support write operations. For example, some shared devices, such as a CD-ROM "jukebox," may be read-only devices. As a result, it is very important that your data handler correctly report its write capabilities to client programs.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

DataHPreextend64

Provides a 64-bit version of DataHPreextend.

```
ComponentResult DataHPreextend64 (
   DataHandler dh,
   const wide *maxToAdd,
   wide *spaceAdded
);
```

Parameters

dh

Identifies the calling program's connection to your data handler component.

maxToAdd

The amount of space to add to the current data reference, in bytes. If the client program sets this parameter to 0, your component should add as much space as it can.

spaceAdded

A pointer to a 64-bit integer. Your component returns the number of bytes it was able to add to the data reference, in bytes.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The only difference between this function and `DataHPreextend` (page 801) is that the `spaceAdded` parameter is a 64-bit integer instead of a 32-bit integer.

Special Considerations

New applications should use this function instead of the 32-bit version.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

DataHPutData

Writes data to a component's current data reference.

```
ComponentResult DataHPutData (
   DataHandler dh,
   Handle h,
   long hOffset,
   long *offset,
   long size
);
```

Parameters

dh

Identifies the calling program's connection to your data handler component.

h

The handle that contains the data to be written to the data reference.

hOffset
Identifies the offset into the handle h to the data to be written.

offset
A pointer to a long integer. Your component returns the offset in the data reference at which your component wrote the data.

size
The number of bytes to write.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function provides a high-level write interface. This is a synchronous write operation that only appends data to the end of the current data reference. That is, the client program's execution is blocked until your component returns control from this function, and the client cannot control where the data is written. As a result, most movie-capture clients (for example, Apple's sequence grabber component) use `DataHWrite` (page 817) to write data when creating movies. As is the case with `DataHScheduleData` (page 806), your component calls the application's data-handler completion function when you are done with the write request.

Special Considerations

Note that some data handlers may not support write operations. For example, some shared devices, such as a CD-ROM "jukebox," may be read-only devices. As a result, it is very important that your data handler correctly report its write capabilities to client programs.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

DataHReadAsync

Undocumented

```
ComponentResult DataHReadAsync (
   DataHandler dh,
   void *dataPtr,
   UInt32 dataSize,
   const wide *dataOffset,
   DataHCompletionUPP completion,
   long refCon
);
```

Parameters

dh
A data handler component.

dataPtr
Undocumented

dataSize
Undocumented

dataOffset
: *Undocumented*

completion
: A pointer to a data-handler completion callback, described in `DataHCompletionProc`.

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

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qtdataref

qtfiletransfer

qtfiletransfer.win

ThreadsImporter

ThreadsImportMovie

Declared In
`QuickTimeComponents.h`

DataHRenameFile

Undocumented

```
ComponentResult DataHRenameFile (
   DataHandler dh,
   Handle newDataRef
);
```

Parameters

dh
: A data handler component.

newDataRef
: A handle to a new QuickTime data reference.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`QuickTimeComponents.h`

DataHResolveDataRef

Locates the container associated with a given data reference.

```
ComponentResult DataHResolveDataRef (
   DataHandler dh,
   Handle theDataRef,
   Boolean *wasChanged,
   Boolean userInterfaceAllowed
);
```

Parameters

dh

Identifies the calling program's connection to your data handler component.

theDataRef

The data reference to be resolved. Different types of containers may require different types of data references. For example, a reference to a memory-based movie may be a handle, while a reference to a file-based movie may be an alias. Apple's memory-based data handler for the Macintosh uses handles (and has a subtype value of `'hndl'`), while the HFS data handler uses Alias Manager aliases (its subtype value is `'alis'`). See `Data References`.

wasChanged

A pointer to a Boolean. Your component should set that Boolean to TRUE if, in locating the container, your data handler updates any information in the data reference.

userInterfaceAllowed

Indicates whether your component may interact with the user when locating the `container`. If this parameter is set to TRUE, your component may ask the user to help locate the container (for instance, by presenting a Find File dialog box).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function permits your component to locate a data reference's container. This function is equivalent to the Mac OS Alias Manager's `ResolveAlias` function. The client program asks your component to locate the container that is associated with a given data reference. If your component determines that the data reference needs to be updated with more accurate location information, it should put the new information in the supplied data reference (and set the Boolean referred to by the `wasChanged` parameter to TRUE). Client programs can correlate data references with data handlers by matching the component's subtype value with the `data` reference type; the subtype value indicates the type of data reference the component supports. All data handlers with the same subtype value must support the same data reference type.

Special Considerations

Client programs may call your data handler's `DataHResolveDataRef` function at any time. Typically, however, the Movie Toolbox uses this function as part of its strategy for opening and reading a movie container. As such, you can expect that the supplied data reference will identify a container that your component can support.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

DataHScheduleData

Reads data from its current data reference, which can be a synchronous read operation or an asynchronous read operation.

```
ComponentResult DataHScheduleData (
   DataHandler dh,
   Ptr PlaceToPutDataPtr,
   long FileOffset,
   long DataSize,
   long RefCon,
   DataHSchedulePtr scheduleRec,
   DataHCompletionUPP CompletionRtn
);
```

Parameters

dh

Identifies the calling program's connection to your data handler component.

PlaceToPutDataPtr

The location in memory that is to receive the data.

FileOffset

The offset in the data reference from which your component is to read.

DataSize

The number of bytes to read.

RefCon

A reference constant that your data handler component should provide to the data-handler completion function specified with the `CompletionRtn` parameter.

scheduleRec

A pointer to a `DataHScheduleRecord`. If this parameter is set to `NIL`, then the client program is requesting a synchronous read operation (that is, your data handler must return the data before returning control to the client program). If this parameter is not set to `NIL`, it must contain the location of a schedule record that has timing information for an asynchronous read request. Your data handler should return control to the client program immediately, and then call the client's data-handler completion function when the data is ready.

CompletionRtn

A pointer to a data handler completion function, described in `DataHCompletionProc`. The client program must provide a completion routine for all asynchronous read requests (that is, all requests that include a valid schedule record). For synchronous requests, client programs should set this parameter to `NIL`. However, if the function is provided, your handler must call it, even after synchronous requests. When your data handler finishes with the client program's read request, your component must call this routine even if the request fails. Your component should pass the reference constant that the client program provided with the `RefCon` parameter.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function provides both a synchronous and an asynchronous read interface. Synchronous read operations work like the `DataHGetData` (page 777) function; the data handler component returns control to the client program only after it has serviced the read request. Asynchronous read operations allow client programs to schedule read requests in the context of a specified QuickTime time base. Your data handler queues the request and immediately returns control to the calling program. After your component actually reads the data, it calls the client program's data-handler completion function. If your component cannot satisfy the request (for example, the request requires data more quickly than you can deliver it), your component should reject the request immediately, rather than queuing the request and then calling the client's data-handler completion function.

```
typedef struct DataHScheduleRecord {
                    TimeRecord timeNeededBy; /* schedule info */
                    long     extendedID; /* type of data */
                    long     extendedVers; /* reserved */
                    Fixed   priority;  /* priority */
                    } DataHScheduleRecord, *DataHSchedulePtr;
```

Special Considerations

Note that the Movie Toolbox may try to read data from a data reference without calling your component's `DataHOpenForRead` (page 796) function. If this happens, your component should open the data reference for read-only access, respond to the read request, and then leave the data reference open in anticipation of later read requests.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

ElectricImageComponent

ElectricImageComponent.win

Declared In

`QuickTimeComponents.h`

DataHScheduleData64

Provides a 64-bit version of DataHScheduleData.

```
ComponentResult DataHScheduleData64 (
   DataHandler dh,
   Ptr PlaceToPutDataPtr,
   const wide *FileOffset,
   long DataSize,
   long RefCon,
   DataHSchedulePtr scheduleRec,
   DataHCompletionUPP CompletionRtn
);
```

Parameters

dh

Identifies the calling program's connection to your data handler component.

PlaceToPutDataPtr
The location in memory that is to receive the data.

FileOffset
A pointer to the offset in the data reference from which your component is to read.

DataSize
The number of bytes to read.

RefCon
A reference constant that your data handler component should provide to the data-handler completion function specified with the `CompletionRtn` parameter.

scheduleRec
A pointer to a `DataHScheduleRecord`. If this parameter is set to `NIL`, then the client program is requesting a synchronous read operation (that is, your data handler must return the data before returning control to the client program). If this parameter is not set to `NIL`, it must contain the location of a schedule record that has timing information for an asynchronous read request. Your data handler should return control to the client program immediately, and then call the client's data-handler completion function when the data is ready.

CompletionRtn
A pointer to a data handler completion function, described in `DataHCompletionProc`. The client program must provide a completion routine for all asynchronous read requests (that is, all requests that include a valid schedule record). For synchronous requests, client programs should set this parameter to `NIL`. However, if the function is provided, your handler must call it, even after synchronous requests. When your data handler finishes with the client program's read request, your component must call this routine even if the request fails. Your component should pass the reference constant that the client program provided with the `RefCon` parameter.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
The only difference between this function and `DataHScheduleData` (page 806) is that the `FileOffset` parameter is a 64-bit integer instead of a 32-bit integer.

Special Considerations

New applications should use this function instead of the 32-bit version.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

DataHSetCacheSizeLimit

Sets the cache size limit for a data handler component.

```
ComponentResult DataHSetCacheSizeLimit (
   DataHandler dh,
   Size cacheSizeLimit
);
```

Parameters

dh

A data handler component.

cacheSizeLimit

A pointer to the cache size limit.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

DataHSetDataRef

Assigns a data reference to your data handler component.

```
ComponentResult DataHSetDataRef (
   DataHandler dh,
   Handle dataRef
);
```

Parameters

dh

Identifies the calling program's connection to your data handler component.

dataRef

The data reference. This parameter contains a handle to the information that identifies the container in question. Different types of containers may require different types of data references. For example, a reference to a memory-based movie may be a handle, while a reference to a file-based movie may be an alias. For example, Apple's memory-based data handler for the Macintosh uses handles (and has a subtype value of `'hndl'`), while the HFS data handler uses Alias Manager aliases (its subtype value is `'alis'`). The client program is responsible for disposing of the handle, so your component must make a copy of it. See `Data References`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function allows your application to assign your data handler's current data reference. All data handler components use data references to identify and locate a movie's container. Client programs can correlate data references with data handlers by matching the component's subtype value with the `data` reference type; the subtype value indicates the type of data reference the component supports. All data handlers with the same subtype value must support the same data reference type.

Special Considerations

Note that the type of data reference always corresponds to the type that your component supports, and that you specify in the `component` subtype value of your data handler. As a result, the client program does not provide a data reference type value (unlike the Movie Toolbox's data reference functions).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

ElectricImageComponent.win

qtdataref

qtfiletransfer

qtfiletransfer.win

ThreadsImportMovie

Declared In

`QuickTimeComponents.h`

DataHSetDataRefExtension

Sets your component's current data reference extension data.

```
ComponentResult DataHSetDataRefExtension (
   DataHandler dh,
   Handle extension,
   OSType idType
);
```

Parameters

dh

A data handler component.

extension

A handle to the extension data.

idType

A four-byte signature identifying the type of extension data.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.1.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

DataHSetDataRefWithAnchor

Sets the data reference and anchor data reference for a data handler.

```
ComponentResult DataHSetDataRefWithAnchor (
   DataHandler dh,
   Handle anchorDataRef,
   OSType dataRefType,
   Handle dataRef
);
```

Parameters

dh

A data handler component.

anchorDataRef

A handle to the anchor data reference.

dataRefType

The type of the data reference. Different types of containers may require different types of data references. For example, a reference to a memory-based movie may be a handle, while a reference to a file-based movie may be an alias. Apple's memory-based data handler for the Macintosh uses handles (and has a subtype value of `'hndl'`), while the HFS data handler uses Alias Manager aliases (its subtype value is `'alis'`). See `Data References`.

dataRef

A data reference handle. Your component should make a copy of its current data reference in a handle and return that handle in this field. The client program is responsible for disposing of that handle.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

DataHSetFileSize

Sets the size, in bytes, of the current data reference.

```
ComponentResult DataHSetFileSize (
   DataHandler dh,
   long fileSize
);
```

Parameters

dh

Identifies the calling program's connection to your data handler component.

fileSize

The new size of the container corresponding to the current data reference, in bytes.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function is operationally equivalent to the Mac OS File Manager's `SetEOF` function. If the client program specifies a new size that is greater than the current size, your component should extend the container to accommodate that new size. If the client program specifies a container size of 0, your component should free all of the space occupied by the container.

Special Considerations

Note that some data handlers may not support write operations. For example, some shared devices, such as a CD-ROM "jukebox," may be read-only devices. As a result, it is very important that your data handler correctly report its write capabilities to client programs.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

DataHSetFileSize64

Provides a 64-bit version of DataHSetFileSize.

```
ComponentResult DataHSetFileSize64 (
   DataHandler dh,
   const wide *fileSize
);
```

Parameters

dh

Identifies the calling program's connection to your data handler component.

fileSize

A pointer to the new size of the container corresponding to the current data reference, in bytes.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
The only difference between this function and `DataHSetFileSize` (page 811) is that the `fileSize` parameter is a 64-bit integer instead of a 32-bit integer.

Special Considerations

New applications should use this function instead of the 32-bit version.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

DataHSetIdleManager

Lets a data handler report its idling needs.

```
ComponentResult DataHSetIdleManager (
   DataHandler dh,
   IdleManager im
);
```

Parameters

dh

A data handler component.

im

A pointer to an opaque data structure that belongs to the Mac OS Idle Manager. You get this pointer by calling `QTIdleManagerOpen` (page 273).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This routine must be implemented by a data handler if the handler needs to report its idling requirements. If you have a handler that supports scheduling reads in the future, you can schedule calls to `DataHTask` (page 816) via the data structure returned by this function.

Version Notes
Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`QuickTimeComponents.h`

DataHSetMacOSFileType

Sets the Mac OS file type for a data handler's current data reference.

```
ComponentResult DataHSetMacOSFileType (
   DataHandler dh,
   OSType fileType
);
```

Parameters

dh

A data handler component.

fileType

A file type; see `File Types and Creators`.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
ElectricImageComponent

ElectricImageComponent.win

Declared In
`QuickTimeComponents.h`

DataHSetMovieUsageFlags

Sets the way that a data handler appends data to its storage.

```
ComponentResult DataHSetMovieUsageFlags (
   DataHandler dh,
   long flags
);
```

Parameters

dh

A data handler component.

flags

Constants (see below) that control data appending. See these constants:

`kDataHMovieUsageDoAppendMDAT`

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`QuickTimeComponents.h`

DataHSetTimeBase

Sets the time base for a data handler component.

```
ComponentResult DataHSetTimeBase (
   DataHandler dh,
   TimeBase tb
);
```

Parameters

dh

A data handler component.

tb

A pointer to a `TimeBaseRecord` structure.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

DataHSetTimeHints

Undocumented

```
ComponentResult DataHSetTimeHints (
   DataHandler dh,
   long flags,
   long bandwidthPriority,
   TimeScale scale,
   TimeValue minTime,
   TimeValue maxTime
);
```

Parameters

dh

A data handler component.

flags

Undocumented

bandwidthPriority

Undocumented

scale

Undocumented

minTime

Undocumented

maxTime

Undocumented

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

DataHTask

Cedes processor time to your data handler.

```
ComponentResult DataHTask (
   DataHandler dh
);
```

Parameters

dh

Identifies the calling program's connection to your data handler component.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function is essentially analogous to the Movie Toolbox's `MoviesTask` (page 257) function. Client programs call this function in order to give your data handler component time to do its work. Because client programs will call this function frequently, and especially so during movie playback or capture, your data handler should return control quickly to the client program.

Special Considerations

Applications should call this function often so that your handler has enough time to do its work.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qtdataref

qtdataref.win

Declared In

`QuickTimeComponents.h`

DataHUpdateMovie

Updates the movie for a data handler's current data reference.

```
ComponentResult DataHUpdateMovie (
   DataHandler dh,
   Movie theMovie,
   short id
);
```

Parameters

dh

A data handler component.

theMovie

A movie identifier. Your application obtains this identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

id

Specifies the resource containing the movie data.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

DataHUseTemporaryDataRef

Undocumented

```
ComponentResult DataHUseTemporaryDataRef (
   DataHandler dh,
   long inFlags
);
```

Parameters

dh

A data handler component.

inFlags

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 6.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`QuickTimeComponents.h`

DataHWrite

Writes data to its current data reference.

```
ComponentResult DataHWrite (
   DataHandler dh,
   Ptr data,
   long offset,
   long size,
   DataHCompletionUPP completion,
   long refCon
);
```

Parameters

dh

Identifies the calling program's connection to your data handler component.

data

Specifies a pointer to the data to be written. Client programs should lock the memory area holding this data, allowing your component's `DataHWrite` function to move memory.

offset

The offset (in bytes) to the location in the current data reference at which to write the data.

size

The number of bytes to write.

completion

A pointer to a data-handler completion function, described in `DataHCompletionProc`. The client program must provide a completion routine for all asynchronous write requests. For synchronous requests, client programs should set this parameter to `NIL`. When your data handler finishes with the client program's write request, your component must call this routine even if the request fails. Your component should pass the reference constant that the client program provided with the `refCon` parameter.

refCon

A reference constant that your data handler component should provide to the data-handler completion function specified with the `completion` parameter. For synchronous operations, client programs should set this parameter to 0.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function provides both a synchronous and an asynchronous write interface. Synchronous write operations work like the `DataHPutData` (page 802) function; the data handler component returns control to the client program only after it has serviced the write request. Asynchronous write operations allow client programs to queue write requests. Your data handler queues the request and immediately returns control to the calling program. After your component actually writes the data, it calls the client program's data-handler completion function.

Special Considerations

Note that some data handlers may not support write operations. For example, some shared devices, such as a CD-ROM "jukebox," may be read-only devices. As a result, it is very important that your data handler correctly report its write capabilities to client programs.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

ElectricImageComponent

ElectricImageComponent.win

qtfiletransfer

qtfiletransfer.win

ThreadsImportMovie

Declared In

`QuickTimeComponents.h`

DataHWrite64

Provides a 64-bit version of DataHWrite.

```
ComponentResult DataHWrite64 (
   DataHandler dh,
   Ptr data,
   const wide *offset,
   long size,
   DataHCompletionUPP completion,
   long refCon
);
```

Parameters

dh

Identifies the calling program's connection to your data handler component.

data

Specifies a pointer to the data to be written. Client programs should lock the memory area holding this data, allowing your component's `DataHWrite` function to move memory.

offset

A pointer to the offset (in bytes) of the location in the current data reference at which to write the data.

size

The number of bytes to write.

completion

A pointer to a data-handler completion function, described in `DataHCompletionProc`. The client program must provide a completion routine for all asynchronous write requests. For synchronous requests, client programs should set this parameter to `NIL`. When your data handler finishes with the client program's write request, your component must call this routine even if the request fails. Your component should pass the reference constant that the client program provided with the `refCon` parameter.

refCon

A reference constant that your data handler component should provide to the data-handler completion function specified with the `completion` parameter. For synchronous operations, client programs should set this parameter to 0.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The only difference between this function and `DataHWrite` (page 817) is that the `offset` parameter is a 64-bit integer instead of a 32-bit integer.

Special Considerations

New applications should use this function instead of the 32-bit version.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

DisposeCDataHandlerUPP

Disposes of a CDataHandlerUPP pointer.

```
void DisposeCDataHandlerUPP (
   CDataHandlerUPP userUPP
);
```

Parameters

`userUPP`

A `CDataHandlerUPP` pointer. See `Universal Procedure Pointers`.

Version Notes

Introduced in QuickTime 6.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`QuickTimeComponents.h`

DisposeCharDataHandlerUPP

Disposes of a CharDataHandlerUPP pointer.

```
void DisposeCharDataHandlerUPP (
   CharDataHandlerUPP userUPP
);
```

Parameters

`userUPP`

A `CharDataHandlerUPP` pointer.

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

DisposeCommentHandlerUPP

Disposes of a CommentHandlerUPP pointer.

```
void DisposeCommentHandlerUPP (
   CommentHandlerUPP userUPP
);
```

Parameters

`userUPP`

A `CommentHandlerUPP` pointer.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

DisposeDataHCompletionUPP

Disposes of a DataHCompletionUPP pointer.

```
void DisposeDataHCompletionUPP (
   DataHCompletionUPP userUPP
);
```

Parameters

`userUPP`

A `DataHCompletionUPP` pointer. See `Universal Procedure Pointers`.

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qtdataref

qtfiletransfer

qtfiletransfer.win

ThreadsImporter

ThreadsImportMovie

Declared In
`QuickTimeComponents.h`

DisposeEndDocumentHandlerUPP

Disposes of an EndDocumentHandlerUPP pointer.

```
void DisposeEndDocumentHandlerUPP (
   EndDocumentHandlerUPP userUPP
);
```

Parameters

userUPP

An `EndDocumentHandlerUPP` pointer.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

DisposeEndElementHandlerUPP

Disposes of an EndElementHandlerUPP pointer.

```
void DisposeEndElementHandlerUPP (
   EndElementHandlerUPP userUPP
);
```

Parameters

userUPP

An `EndElementHandlerUPP` pointer.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

DisposePreprocessInstructionHandlerUPP

Disposes of a PreprocessInstructionHandlerUPP pointer.

```
void DisposePreprocessInstructionHandlerUPP (
   PreprocessInstructionHandlerUPP userUPP
);
```

Parameters

userUPP

A `PreprocessInstructionHandlerUPP` pointer.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

DisposeStartDocumentHandlerUPP

Disposes of a StartDocumentHandlerUPP pointer.

```
void DisposeStartDocumentHandlerUPP (
   StartDocumentHandlerUPP userUPP
);
```

Parameters

userUPP

A `StartDocumentHandlerUPP` pointer.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

DisposeStartElementHandlerUPP

Disposes of a StartElementHandlerUPP pointer.

```
void DisposeStartElementHandlerUPP (
   StartElementHandlerUPP userUPP
);
```

Parameters

userUPP

A `StartElementHandlerUPP` pointer.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

DisposeVdigIntUPP

Disposes of a VdigIntUPP pointer.

```
void DisposeVdigIntUPP (
   VdigIntUPP userUPP
);
```

Parameters

`userUPP`
A `VdigIntUPP` pointer. See `Universal Procedure Pointers`.

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

NewCDataHandlerUPP

Allocates a Universal Procedure Pointer for the CDataHandlerProc callback.

```
CDataHandlerUPP NewCDataHandlerUPP (
   CDataHandler userRoutine
);
```

Parameters

`userRoutine`
Undocumented

Return Value
A new UPP; see `Universal Procedure Pointers`.

Version Notes
Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`QuickTimeComponents.h`

NewCharDataHandlerUPP

Undocumented

```
CharDataHandlerUPP NewCharDataHandlerUPP (
   CharDataHandler userRoutine
);
```

Parameters

userRoutine

Undocumented

Return Value
A new UPP; see `Universal Procedure Pointers`.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

NewCommentHandlerUPP

Undocumented

```
CommentHandlerUPP NewCommentHandlerUPP (
   CommentHandler userRoutine
);
```

Parameters

userRoutine

Undocumented

Return Value
A new UPP; see `Universal Procedure Pointers`.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

NewDataHCompletionUPP

Allocates a Universal Procedure Pointer for the DataHCompletionProc callback.

```
DataHCompletionUPP NewDataHCompletionUPP (
   DataHCompletionProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your application-defined function.

Return Value

A new UPP; see `Universal Procedure Pointers`.

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes

Introduced in QuickTime 4.1. Replaces `NewDataHCompletionProc`.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qtdataref

qtfiletransfer

qtfiletransfer.win

ThreadsImporter

ThreadsImportMovie

Declared In

`QuickTimeComponents.h`

NewEndDocumentHandlerUPP

Undocumented

```
EndDocumentHandlerUPP NewEndDocumentHandlerUPP (
   EndDocumentHandler userRoutine
);
```

Parameters

userRoutine

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

NewEndElementHandlerUPP

Undocumented

```
EndElementHandlerUPP NewEndElementHandlerUPP (
   EndElementHandler userRoutine
);
```

Parameters

userRoutine

Undocumented

Return Value
A new UPP; see `Universal Procedure Pointers`.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

NewPreprocessInstructionHandlerUPP

Undocumented

```
PreprocessInstructionHandlerUPP NewPreprocessInstructionHandlerUPP (
   PreprocessInstructionHandler userRoutine
);
```

Parameters

userRoutine

Undocumented

Return Value
A new UPP; see `Universal Procedure Pointers`.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

NewStartDocumentHandlerUPP

Undocumented

```
StartDocumentHandlerUPP NewStartDocumentHandlerUPP (
   StartDocumentHandler userRoutine
);
```

Parameters

userRoutine

Undocumented

Return Value
A new UPP; see `Universal Procedure Pointers`.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

NewStartElementHandlerUPP

Undocumented

```
StartElementHandlerUPP NewStartElementHandlerUPP (
   StartElementHandler userRoutine
);
```

Parameters

userRoutine

Undocumented

Return Value
A new UPP; see `Universal Procedure Pointers`.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

NewVdigIntUPP

Allocates a Universal Procedure Pointer for the VdigIntProc callback.

```
VdigIntUPP NewVdigIntUPP (
   VdigIntProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your application-defined function.

Return Value
A new UPP; see `Universal Procedure Pointers`.

Discussion
This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes
Introduced in QuickTime 4.1. Replaces `NewVdigIntProc`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

XMLParseAddAttribute

Undocumented

```
ComponentResult XMLParseAddAttribute (
   ComponentInstance aParser,
   UInt32 elementID,
   UInt32 nameSpaceID,
   char *attributeName,
   UInt32 *attributeID
);
```

Parameters

aParser
: *Undocumented*

elementID
: *Undocumented*

nameSpaceID
: *Undocumented*

attributeName
: *Undocumented*

attributeID
: *Undocumented*

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

XMLParseAddAttributeAndValue

Undocumented

```
ComponentResult XMLParseAddAttributeAndValue (
   ComponentInstance aParser,
   UInt32 elementID,
   UInt32 nameSpaceID,
   char *attributeName,
   UInt32 *attributeID,
   UInt32 attributeValueKind,
   void *attributeValueKindInfo
);
```

Parameters

aParser

    Undocumented

elementID

    Undocumented

nameSpaceID

    Undocumented

attributeName

    Undocumented

attributeID

    Undocumented

attributeValueKind

    Undocumented

attributeValueKindInfo

    Undocumented

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

XMLParseAddAttributeValueKind

Undocumented

```
ComponentResult XMLParseAddAttributeValueKind (
   ComponentInstance aParser,
   UInt32 elementID,
   UInt32 attributeID,
   UInt32 attributeValueKind,
   void *attributeValueKindInfo
);
```

Parameters

aParser
: *Undocumented*

elementID
: *Undocumented*

attributeID
: *Undocumented*

attributeValueKind
: *Undocumented*

attributeValueKindInfo
: *Undocumented*

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

XMLParseAddElement

Undocumented

```
ComponentResult XMLParseAddElement (
   ComponentInstance aParser,
   char *elementName,
   UInt32 nameSpaceID,
   UInt32 *elementID,
   long elementFlags
);
```

Parameters

aParser
: *Undocumented*

elementName
: *Undocumented*

nameSpaceID
: *Undocumented*

elementID

Undocumented

elementFlags

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

XMLParseAddMultipleAttributes

Undocumented

```
ComponentResult XMLParseAddMultipleAttributes (
   ComponentInstance aParser,
   UInt32 elementID,
   UInt32 *nameSpaceIDs,
   char *attributeNames,
   UInt32 *attributeIDs
);
```

Parameters

aParser

Undocumented

elementID

Undocumented

nameSpaceIDs

Undocumented

attributeNames

Undocumented

attributeIDs

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

XMLParseAddMultipleAttributesAndValues

Undocumented

```
ComponentResult XMLParseAddMultipleAttributesAndValues (
   ComponentInstance aParser,
   UInt32 elementID,
   UInt32 *nameSpaceIDs,
   char *attributeNames,
   UInt32 *attributeIDs,
   UInt32 *attributeValueKinds,
   void **attributeValueKindInfos
);
```

Parameters

aParser
    Undocumented

elementID
    Undocumented

nameSpaceIDs
    Undocumented

attributeNames
    Undocumented

attributeIDs
    Undocumented

attributeValueKinds
    Undocumented

attributeValueKindInfos
    Undocumented

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

XMLParseAddNameSpace

Undocumented

```
ComponentResult XMLParseAddNameSpace (
   ComponentInstance aParser,
   char *nameSpaceURL,
   UInt32 *nameSpaceID
);
```

Parameters

aParser

Undocumented

nameSpaceURL

Undocumented

nameSpaceID

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

XMLParseDataRef

Undocumented

```
ComponentResult XMLParseDataRef (
   ComponentInstance aParser,
   Handle dataRef,
   OSType dataRefType,
   long parseFlags,
   XMLDoc *document
);
```

Parameters

aParser

Undocumented

dataRef

Undocumented

dataRefType

Undocumented

parseFlags

Undocumented

document

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

XMLParseDisposeXMLDoc

Undocumented

```
ComponentResult XMLParseDisposeXMLDoc (
   ComponentInstance aParser,
   XMLDoc document
);
```

Parameters

aParser
Undocumented

document
Undocumented

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

XMLParseFile

Undocumented

```
ComponentResult XMLParseFile (
   ComponentInstance aParser,
   ConstFSSpecPtr fileSpec,
   long parseFlags,
   XMLDoc *document
);
```

Parameters

aParser
Undocumented

fileSpec
Undocumented

parseFlags

Undocumented

document

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

XMLParseGetDetailedParseError

Undocumented

```
ComponentResult XMLParseGetDetailedParseError (
   ComponentInstance aParser,
   long *errorLine,
   StringPtr errDesc
);
```

Parameters

aParser

Undocumented

errorLine

Undocumented

errDesc

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

XMLParseSetCDataHandler

Undocumented

```
ComponentResult XMLParseSetCDataHandler (
   ComponentInstance aParser,
   CDataHandlerUPP cdata
);
```

Parameters

aParser

  Undocumented

cdata

  Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 6.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`QuickTimeComponents.h`

XMLParseSetCharDataHandler

Undocumented

```
ComponentResult XMLParseSetCharDataHandler (
   ComponentInstance aParser,
   CharDataHandlerUPP charData
);
```

Parameters

aParser

  Undocumented

charData

  Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

XMLParseSetCommentHandler

Undocumented

```
ComponentResult XMLParseSetCommentHandler (
   ComponentInstance aParser,
   CommentHandlerUPP comment
);
```

Parameters

aParser
  Undocumented

comment
  Undocumented

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

XMLParseSetEndDocumentHandler

Undocumented

```
ComponentResult XMLParseSetEndDocumentHandler (
   ComponentInstance aParser,
   EndDocumentHandlerUPP endDocument
);
```

Parameters

aParser
  Undocumented

endDocument
  Undocumented

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

XMLParseSetEndElementHandler

Undocumented

```
ComponentResult XMLParseSetEndElementHandler (
   ComponentInstance aParser,
   EndElementHandlerUPP endElement
);
```

Parameters

aParser

Undocumented

endElement

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

XMLParseSetEventParseRefCon

Undocumented

```
ComponentResult XMLParseSetEventParseRefCon (
   ComponentInstance aParser,
   long refcon
);
```

Parameters

aParser

Undocumented

refcon

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

XMLParseSetOffsetAndLimit

Undocumented

```
ComponentResult XMLParseSetOffsetAndLimit (
   ComponentInstance aParser,
   UInt32 offset,
   UInt32 limit
);
```

Parameters

aParser

    Undocumented

offset

    Undocumented

limit

    Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

XMLParseSetPreprocessInstructionHandler

Undocumented

```
ComponentResult XMLParseSetPreprocessInstructionHandler (
   ComponentInstance aParser,
   PreprocessInstructionHandlerUPP preprocessInstruction
);
```

Parameters

aParser

    Undocumented

preprocessInstruction

    Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

XMLParseSetStartDocumentHandler

Undocumented

```
ComponentResult XMLParseSetStartDocumentHandler (
   ComponentInstance aParser,
   StartDocumentHandlerUPP startDocument
);
```

Parameters

aParser

 Undocumented

startDocument

 Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

XMLParseSetStartElementHandler

Undocumented

```
ComponentResult XMLParseSetStartElementHandler (
   ComponentInstance aParser,
   StartElementHandlerUPP startElement
);
```

Parameters

aParser

 Undocumented

startElement

 Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

Callbacks

DataHCompletionProc

Called upon completion of a read or write operation.

```
typedef void (*DataHCompletionProcPtr) (Ptr request, long refcon, OSErr err);
```

If you name your function `MyDataHCompletionProc`, you would declare it this way:

```
void MyDataHCompletionProc (
    Ptr       request,
    long      refcon,
    OSErr     err );
```

Parameters

request

Specifies a pointer to the data that was associated with the read request `DataHScheduleData` (page 806) or write request `DataHWrite` (page 817). The client program uses this pointer to determine which request has completed.

refcon

A reference constant that the client program supplied to your data handler component when it made the original request.

err

Indicates the success or failure of the operation. If the operation succeeded, set this parameter to 0. Otherwise, specify an appropriate error code.

Discussion

Data handler completion functions are guaranteed to be called at non-interrupt time. This means that you can safely call functions that are not interrupt-safe, such as `DataHScheduleData` (page 806), from within a completion function.

Declared In

`QuickTimeComponents.h`

Data Types

ConstFSSpecPtr

Represents a type used by the Data Components API.

```
typedef const FSSpec * ConstFSSpecPtr;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Files.h`

DataCodecComponent

Represents a type used by the Data Components API.

```
typedef ComponentInstance DataCodecComponent;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

DataHCompletionUPP

Represents a type used by the Data Components API.

```
typedef STACK_UPP_TYPE(DataHCompletionProcPtr) DataHCompletionUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

DataHFileTypeOrderingHandle

Represents a type used by the Data Components API.

```
typedef DataHFileTypeOrderingPtr * DataHFileTypeOrderingHandle;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

DataHFileTypeOrderingPtr

Represents a type used by the Data Components API.

```
typedef OSType * DataHFileTypeOrderingPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

DataHSchedulePtr

Represents a type used by the Data Components API.

```
typedef DataHScheduleRecord * DataHSchedulePtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

DataHScheduleRecord

Provides scheduling information for scheduled reads.

```
struct DataHScheduleRecord {
     TimeRecord    timeNeededBy;
     long          extendedID;
     long          extendedVers;
     Fixed         priority;
 };
```

Fields
`timeNeededBy`

Discussion
Specifies the time at which your data handler must deliver the requested data to the calling program. This time value is relative to the time base that is contained in this time record. During pre-roll operations, the Movie Toolbox may use special values in certain time record fields. The time record fields in question are the `scale` and value fields. By correctly interpreting the `values` of these fields, your data handler can queue up the pre-roll read requests in the most efficient way for its device.

`extendedID`

Discussion
A constant (see below) that indicates the type of data that follows in the remainder of the structure. See these constants:
`kDataHExtendedSchedule`

`extendedVers`

Discussion
Reserved. This field should always be set to 0.

`priority`

Discussion
Indicates the relative importance of the data request. Client programs assign a value of 100.0 to data requests the must be delivered. Lower values indicate relatively less critical data. If your data handler must accommodate bandwidth limitations when delivering data, your component may use this value as an indication of which requests can be dropped with the least impact on the client program. As an example, consider using priorities in a frame-differenced movie. Key frames might have priority values of 100.0, indicating that they are essential to proper playback. As you move through the frames following a key frame, each successive frame might have a lower priority value. Once you drop a frame, you must drop all successive frames of equal or lower priority until you reach another key frame, because each of these frames would rely on the dropped one for some image data.

Discussion
There are two types of preroll read operations. The first type is a required read; that is, the Movie Toolbox requires that the read operation be satisfied before the movie starts playing. The second type is an optional read. If your data handler can satisfy the read operation as part of the pre-roll operation, it should do so.

Otherwise, your data handler may satisfy the request at a specified time while the movie is playing. The Movie Toolbox indicates that a preroll read request is required by setting the `scale` field of the time record to -1. This literally means that the request is scheduled for a time that is infinitely far into the future. Your data handler should collect all such read requests, order them most efficiently for your device, and process them when the Movie Toolbox calls your component's `DataHFinishData` (page 774) function. For optional preroll read requests, the Movie Toolbox sets the `scale` field properly, but negates the `contents` of the `value` field. Your data handler has the option of delivering the data for this request with the required data, if that can be done efficiently. Otherwise, your data handler may deliver the data at its schedule time. You determine the scheduled time by negating the `contents` of the `value` field (that is, multiplying by -1).

Declared In
`QuickTimeComponents.h`

DataHVolumeList

Represents a type used by the Data Components API.

```
typedef DataHVolumeListPtr * DataHVolumeList;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

DataHVolumeListPtr

Represents a type used by the Data Components API.

```
typedef DataHVolumeListRecord * DataHVolumeListPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

XMLDoc

Represents a type used by the Data Components API.

```
typedef XMLDocRecord * XMLDoc;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

XMLDocRecord

Undocumented

```
struct XMLDocRecord {
     void *        xmlDataStorage;
     XMLElement    rootElement;
 };
```

Fields

`xmlDataStorage`

Discussion

Undocumented

`rootElement`

Discussion

Undocumented

Declared In

`QuickTimeComponents.h`

Constants

kDataHCanRead

Constants grouped with kDataHCanRead.

```
enum {
  kDataHCanRead                = 1L << 0,
  kDataHSpecialRead            = 1L << 1,
  kDataHSpecialReadFile        = 1L << 2,
  kDataHCanWrite               = 1L << 3,
  kDataHSpecialWrite           = 1 << 4,
  kDataHSpecialWriteFile       = 1 << 5,
  kDataHCanStreamingWrite      = 1 << 6,
  kDataHMustCheckDataRef       = 1 << 7
};
```

Constants

`kDataHCanRead`

Indicates that your data handler can read from the volume.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeComponents.h`.

`kDataHSpecialRead`

Indicates that your data handler can read from the volume using a specialized method. For example, your data handler might support access to networked multimedia servers using a special protocol. In that case, your component would set this flag to 1 whenever the volume resides on a supported server.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeComponents.h`.

`kDataHSpecialReadFile`
: Reserved for use by Apple.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`kDataHCanWrite`
: Indicates that your data handler can write data to the volume. In particular, use this flag to indicate that your data handler's `DataHPutData` (page 802) function will work with this volume.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`kDataHSpecialWrite`
: Indicates that your data handler can write to the volume using a specialized method. As with the `kDataHSpecialRead` flag, your data handler would use this flag to indicate that your component can access the volume using specialized support (for example, special network protocols).

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`kDataHCanStreamingWrite`
: Indicates that your data handler can support the special write functions for capturing movie data when writing to this volume.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

Declared In
`QuickTimeComponents.h`

DataHScheduleRecord Values

Constants passed to DataHScheduleRecord.

```
enum {
  kDataHExtendedSchedule        = 'xtnd'
};
```

Declared In
`QuickTimeComponents.h`

DataHGetFileTypeOrdering Values

Constants passed to DataHGetFileTypeOrdering.

```
enum {
  kDataHFileTypeMacOSFileType   = 'ftyp',
  kDataHFileTypeExtension       = 'fext',
  kDataHFileTypeMIME            = 'mime'
};
```

Declared In
`QuickTimeComponents.h`

DataHGetInfoFlags Values

Constants passed to DataHGetInfoFlags.

```
enum {
   kDataHInfoFlagNeverStreams    = 1 << 0, /* set if this data handler doesn't
stream*/
  kDataHInfoFlagCanUpdateDataRefs = 1 << 1, /* set if this data handler might update
 data reference*/
  kDataHInfoFlagNeedsNetworkBandwidth = 1 << 2 /* set if this data handler may need
 to occupy the network*/
};
```

Declared In
QuickTimeComponents.h

DataHSetMovieUsageFlags Values

Constants passed to DataHSetMovieUsageFlags.

```
enum {
  kDataHMovieUsageDoAppendMDAT  = 1L << 0 /* if set, datahandler should append wide
 and mdat atoms in append call*/
};
```

Declared In
QuickTimeComponents.h

Image Codec Reference for QuickTime

Framework:	Frameworks/QuickTime.framework
Declared in	ImageCodec.h

Overview

An image codec (or image compressor component) is a code resource that provides QuickTime with compression or decompression services for image data.

Functions by Task

Base Image Decompressor Functions

`ImageCodecBeginBand` (page 869)
: Called before drawing a band or frame; it allows your image decompressor component to save information about a band before decompressing it.

`ImageCodecDisposeMemory` (page 876)
: Disposes codec-allocated memory.

`ImageCodecDITLEvent` (page 877)
: Lets an image codec component receive and process dialog events.

`ImageCodecDITLInstall` (page 878)
: Installs added items in an image codec settings dialog box before the dialog box is displayed to the user.

`ImageCodecDITLItem` (page 878)
: Receives and processes mouse clicks in the image codec settings dialog box.

`ImageCodecDITLRemove` (page 879)
: Removes a panel from the image codec settings dialog box.

`ImageCodecDITLValidateInput` (page 880)
: Validates the contents of the user dialog box for an image codec component.

`ImageCodecDrawBand` (page 880)
: Decompresses a band or frame.

`ImageCodecEndBand` (page 888)
: Notifies your image decompressor component that decompression of a band has finished or that it was terminated by the Image Compression Manager.

`ImageCodecExtractAndCombineFields` (page 889)
: Performs field operations on video data.

`ImageCodecFlush` (page 891)
: Empties an image decompressor component's input queue of pending scheduled frames.

`ImageCodecGetDITLForSize` (page 897)
: Returns the size of various dialog item lists.

`ImageCodecGetMaxCompressionSizeWithSources` (page 899)
: Notifies your codec when an application calls GetCSequenceMaxCompressionSize.

`ImageCodecGetSettings` (page 901)
: Returns the codec settings chosen by the user.

`ImageCodecHitTestData` (page 904)
: Notifies your codec when the application calls PtInDSequenceData.

`ImageCodecInitialize` (page 906)
: Called before making any other all calls to your component.

`ImageCodecIsImageDescriptionEquivalent` (page 906)
: Compares image descriptions.

`ImageCodecMergeFloatingImageOntoWindow` (page 908)
: Draws the current contents of a floating image.

`ImageCodecNewImageBufferMemory` (page 909)
: Asks a codec to allocate memory for an offscreen buffer of non-RGB pixels.

`ImageCodecNewMemory` (page 911)
: Requests codec-allocated memory.

`ImageCodecPreflight` (page 913)
: Called before decompressing an image, in response to an ImageCodecPreDecompress call from the Image Compression Manager.

`ImageCodecQueueStarting` (page 916)
: Called by the base image decompressor before decompressing the frames in the queue if your image decompressor component supports asynchronous scheduled decompression.

`ImageCodecQueueStopping` (page 916)
: Notifies your component that the frames in the queue have been decompressed, if your image decompressor component supports asynchronous scheduled decompression.

`ImageCodecRemoveFloatingImage` (page 917)
: Hides an image codec's floating image without having to close the component.

`ImageCodecRequestSettings` (page 919)
: Displays a dialog box containing codec-specific compression settings.

`ImageCodecSetSettings` (page 920)
: Sets the settings of an optional image codec dialog box.

`ImageCodecSetTimeCode` (page 922)
: Sets the timecode for the next frame that is to be decompressed.

`ImageCodecSourceChanged` (page 922)
: Notifies your codec that one of the data sources has changed when an application calls CDSequenceSetSourceData or CDSequenceChangedSourceData.

Low-Level Effects Functions

`ImageCodecCreateStandardParameterDialog` (page 873)
Creates a parameters dialog box for a specified effect.

`ImageCodecDismissStandardParameterDialog` (page 875)
Retrieves values from a standard parameter dialog box created by the low-level ImageCodecCreateStandardParameterDialog function, then closes the dialog box.

`ImageCodecGetParameterList` (page 900)
Returns a parameter description atom container for a specified effect component instance.

`ImageCodecIsStandardParameterDialogEvent` (page 907)
Processes events related to a standard parameters dialog box created by ImageCodecCreateStandardParameterDialog.

`ImageCodecStandardParameterDialogDoAction` (page 923)
Allows you to control the behavior of a standard parameter dialog box created by ImageCodecCreateStandardParameterDialog.

Vector Codec Component Functions

`CurveAddAtomToVectorStream` (page 854)
Adds an atom to a vector data stream.

`CurveAddPathAtomToVectorStream` (page 855)
Adds a path to a vector data stream.

`CurveAddZeroAtomToVectorStream` (page 856)
Adds a kCurveEndAtom to a vector data stream; this atom marks the end of the vector data stream,

`CurveCountPointsInPath` (page 856)
Counts the points along either one of a path's contours or all of its contours.

`CurveCreateVectorStream` (page 857)
Creates a new, empty vector data stream.

`CurveGetAtomDataFromVectorStream` (page 858)
Finds the first atom of a specified type within a vector data stream and get its data.

`CurveGetLength` (page 859)
Calculates the length of one of a path's contours or the sum of the lengths of all of its contours.

`CurveGetNearestPathPoint` (page 859)
Finds the closest point on a path to a specified point.

`CurveGetPathPoint` (page 860)
Obtains a point from a path and to find out if the point is on the curve.

`CurveInsertPointIntoPath` (page 861)
Adds a new point to a path.

`CurveLengthToPoint` (page 862)
Obtains the point at a specified distance along a curve.

`CurveNewPath` (page 863)
Creates a new path.

`CurvePathPointToLength` (page 864)
Obtains the length of a path between specified starting and ending distances that is nearest a point.

`CurveSetPathPoint` (page 865)
: Changes the location of a point in a path.

Supporting Functions

`DisposeImageCodecDrawBandCompleteUPP` (page 866)
: Disposes of an ImageCodecDrawBandCompleteUPP pointer.

`DisposeImageCodecMPDrawBandUPP` (page 866)
: Disposes of an ImageCodecMPDrawBandUPP pointer.

`DisposeImageCodecTimeTriggerUPP` (page 867)
: Disposes of an ImageCodecTimeTriggerUPP pointer.

`ImageCodecBandCompress` (page 867)
: Asks your component to compress an image or a band of an image.

`ImageCodecBandDecompress` (page 868)
: Asks your component to decompress a frame.

`ImageCodecBeginPass` (page 870)
: Notifies the compressor that it should operate in multipass mode and use the given multipass storage.

`ImageCodecBusy` (page 871)
: Lets your component report whether it is performing an asynchronous operation.

`ImageCodecCancelTrigger` (page 872)
: Cancels an image codec's ImageCodecTimeTriggerProc callback.

`ImageCodecCompleteFrame` (page 872)
: Directs the compressor to finish with a queued source frame, either emitting or dropping it.

`ImageCodecDecodeBand` (page 874)
: Returns an ImageSubCodecDecompressRecord structure for an image codec component.

`ImageCodecDisposeImageGWorld` (page 876)
: Disposes of an image graphics world associated with an image codec.

`ImageCodecDroppingFrame` (page 881)
: Undocumented

`ImageCodecEffectBegin` (page 882)
: Undocumented

`ImageCodecEffectCancel` (page 882)
: Undocumented

`ImageCodecEffectConvertEffectSourceToFormat` (page 883)
: Undocumented

`ImageCodecEffectDisposeSMPTEFrame` (page 884)
: Undocumented

`ImageCodecEffectGetSpeed` (page 884)
: Undocumented

`ImageCodecEffectPrepareSMPTEFrame` (page 885)
: Undocumented

`ImageCodecEffectRenderFrame` (page 885)
: Undocumented

`ImageCodecEffectRenderSMPTEFrame` (page 886)
Undocumented

`ImageCodecEffectSetup` (page 887)
Undocumented

`ImageCodecEncodeFrame` (page 888)
Presents the compressor with a frame to encode.

`ImageCodecFlushFrame` (page 892)
Undocumented

`ImageCodecGetBaseMPWorkFunction` (page 892)
Gets an image codec's ComponentMPWorkFunctionProc callback.

`ImageCodecGetCodecInfo` (page 893)
Notifies your codec whenever an application calls GetCodecInfo.

`ImageCodecGetCompressedImageSize` (page 894)
Notifies your codec whenever an application calls GetCompressedImageSize.

`ImageCodecGetCompressionTime` (page 895)
Notifies your codec whenever an application calls GetCompressionTime.

`ImageCodecGetDecompressLatency` (page 896)
Retrieves the video latency value from a specified video codec.

`ImageCodecGetMaxCompressionSize` (page 898)
Notifies your codec whenever an application calls GetMaxCompressionSize.

`ImageCodecGetParameterListHandle` (page 901)
Returns a handle to a Mac OS resource of type 'atms'.

`ImageCodecGetSettingsAsText` (page 902)
Undocumented

`ImageCodecGetSimilarity` (page 902)
Notifies your codec when an application calls GetSimilarity.

`ImageCodecGetSourceDataGammaLevel` (page 903)
Returns the native gamma of compressed data, if any.

`ImageCodecHitTestDataWithFlags` (page 905)
Undocumented

`ImageCodecNewImageGWorld` (page 910)
Undocumented

`ImageCodecPreCompress` (page 912)
Notifies your component before compressing an image or a band of an image.

`ImageCodecPreDecompress` (page 912)
Notifies your component before decompressing an image or sequence of frames.

`ImageCodecPrepareToCompressFrames` (page 914)
Prepares the compressor to receive frames.

`ImageCodecProcessBetweenPasses` (page 915)
Provides the compressor with an opportunity to perform processing between passes.

`ImageCodecRequestGammaLevel` (page 918)
Asks an image codec to convert from source to destination gamma levels.

`ImageCodecScheduleFrame` (page 920)
Undocumented

ImageCodecSetTimeBase (page 921)
: Sets the time base for an image codec component.

ImageCodecTrimImage (page 925)
: Notifies your component whenever an application calls TrimImage.

ImageCodecValidateParameters (page 926)
: Validates effect parameters.

NewImageCodecDrawBandCompleteUPP (page 927)
: Allocates a Universal Procedure Pointer for an ImageCodecDrawBandCompleteProc callback.

NewImageCodecMPDrawBandUPP (page 927)
: Allocates a Universal Procedure Pointer for the ImageCodecMPDrawBandProc callback.

NewImageCodecTimeTriggerUPP (page 928)
: Allocates a Universal Procedure Pointer for the ImageCodecTimeTriggerProc callback.

QTPhotoDefineHuffmanTable (page 928)
: Defines a Huffman table.

QTPhotoDefineQuantizationTable (page 929)
: Specifies a custom quantization table.

QTPhotoSetRestartInterval (page 930)
: Specifies the restart interval to use in future JPEG compression operations.

QTPhotoSetSampling (page 931)
: Specifies the chrominance downsampling ratio to use in future JPEG compression operations.

Functions

CurveAddAtomToVectorStream

Adds an atom to a vector data stream.

```
ComponentResult CurveAddAtomToVectorStream (
   ComponentInstance effect,
   OSType atomType,
   Size atomSize,
   void *pAtomData,
   Handle vectorStream
);
```

Parameters

effect
: The instance of the QuickTime vector codec component for the request. Your software obtains this reference when calling the Component Manager's OpenComponent or OpenDefaultComponent function.

atomType
: The type of atom to add to the vector data stream.

atomSize
: The size of the data for the atom.

pAtomData
: A pointer to the data for the atom.

`vectorStream`
: A handle to the vector data stream to which to add the atom.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function adds the atom to the end of the specified vector data stream and resizes the vector data stream handle.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qtvectors

qtvectors.win

Declared In
`ImageCodec.h`

CurveAddPathAtomToVectorStream

Adds a path to a vector data stream.

```
ComponentResult CurveAddPathAtomToVectorStream (
   ComponentInstance effect,
   Handle pathData,
   Handle vectorStream
);
```

Parameters

effect
: The instance of the QuickTime vector codec component for the request. Your software obtains this reference when calling the Component Manager's `OpenComponent` or `OpenDefaultComponent` function.

pathData
: A handle to the data for the path.

vectorStream
: A handle to the vector data stream to which to add the path.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function adds the path to the end of the specified vector data stream and resizes the vector data stream handle.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code

qtvectors

qtvectors.win

Declared In

`ImageCodec.h`

CurveAddZeroAtomToVectorStream

Adds a kCurveEndAtom to a vector data stream; this atom marks the end of the vector data stream,

```
ComponentResult CurveAddZeroAtomToVectorStream (
   ComponentInstance effect,
   Handle vectorStream
);
```

Parameters

`effect`

The instance of the QuickTime vector codec component for the request. Your software obtains this reference when calling the Component Manager's `OpenComponent` or `OpenDefaultComponent` function.

`vectorStream`

A handle to the vector data stream to which to add the `kCurveEndAtom` atom.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function adds a `kCurveEndAtom` atom to the end of the specified vector data stream and resizes the vector data stream handle. The `kCurveEndAtom` atom is required at the end of a vector data stream, and there may be only one `kCurveEndAtom` atom in the stream.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qtvectors

qtvectors.win

Declared In

`ImageCodec.h`

CurveCountPointsInPath

Counts the points along either one of a path's contours or all of its contours.

```
ComponentResult CurveCountPointsInPath (
   ComponentInstance effect,
   gxPaths *aPath,
   unsigned long contourIndex,
   unsigned long *pCount
);
```

Parameters

effect
: The instance of the QuickTime vector codec component for the request.

aPath
: A pointer to the path.

contourIndex
: The index of the contour to be counted.

pCount
: A pointer to a field that is to receive the number of points in the contour or path.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCodec.h`

CurveCreateVectorStream

Creates a new, empty vector data stream.

```
ComponentResult CurveCreateVectorStream (
   ComponentInstance effect,
   Handle *pStream
);
```

Parameters

effect
: The instance of the QuickTime vector codec component for the request. Your software obtains this reference when calling the Component Manager's `OpenComponent` or `OpenDefaultComponent` function.

pStream
: A pointer to the handle that is to receive the newly created vector data stream.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
The caller is responsible for disposing of the stream when finished with it. This can be done by calling `DisposeHandle`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qtvectors

qtvectors.win

Declared In
`ImageCodec.h`

CurveGetAtomDataFromVectorStream

Finds the first atom of a specified type within a vector data stream and get its data.

```
ComponentResult CurveGetAtomDataFromVectorStream (
   ComponentInstance effect,
   Handle vectorStream,
   long atomType,
   long *dataSize,
   Ptr *dataPtr
);
```

Parameters

`effect`
: The instance of the QuickTime vector codec component for the request. Your software obtains this reference when calling the Component Manager's `OpenComponent` or `OpenDefaultComponent` function.

`vectorStream`
: A handle to the vector data stream from which to get the atom.

`atomType`
: The type of atom to find.

`dataSize`
: A pointer to a field that is to receive the size of the atom's data.

`dataPtr`
: A pointer to a pointer to a field that is to receive the atom's data.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Before calling this function, your software must lock the handle for the vector data stream (with Macintosh, by calling `HLock`). This prevents the handle from being moved, which could invalidate the pointer to the atom data before your software gets the data.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCodec.h`

CurveGetLength

Calculates the length of one of a path's contours or the sum of the lengths of all of its contours.

```
ComponentResult CurveGetLength (
   ComponentInstance effect,
   gxPaths *target,
   long index,
   wide *wideLength
);
```

Parameters

`effect`
The instance of the QuickTime vector codec component for the request. Your software obtains this reference when calling the Component Manager's `OpenComponent` or `OpenDefaultComponent` function.

`target`
A pointer to the path.

`index`
Contains the index of the contour whose length is to be calculated or, if the value is 0, specifies to calculate the lengths of all of the path's contours and return the sum of the lengths.

`wideLength`
A pointer to a field that is to receive the length.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCodec.h`

CurveGetNearestPathPoint

Finds the closest point on a path to a specified point.

```
ComponentResult CurveGetNearestPathPoint (
   ComponentInstance effect,
   gxPaths *aPath,
   FixedPoint *thePoint,
   unsigned long *contourIndex,
   unsigned long *pointIndex,
   Fixed *theDelta
);
```

Parameters

effect
: The instance of the QuickTime vector codec component for the request. Your software obtains this reference when calling the Component Manager's `OpenComponent` or `OpenDefaultComponent` function.

aPath
: A pointer to the path.

thePoint
: A pointer to a point for which to find the closest point on the path.

contourIndex
: A pointer to a field that is to receive the index of the contour that contains the closest point.

pointIndex
: A pointer to a field that is to receive the index of the closest point.

theDelta
: A pointer to a field that is to receive the distance between the specified point and the closest point in the contour to it.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

In programs where users directly manipulate curves, you can use this function to determine the closest control point to a given point.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCodec.h`

CurveGetPathPoint

Obtains a point from a path and to find out if the point is on the curve.

```
ComponentResult CurveGetPathPoint (
   ComponentInstance effect,
   gxPaths *aPath,
   unsigned long contourIndex,
   unsigned long pointIndex,
   gxPoint *thePoint,
   Boolean *ptIsOnPath
);
```

Parameters

effect

The instance of the QuickTime vector codec component for the request. Your software obtains this reference when calling the Component Manager's `OpenComponent` or `OpenDefaultComponent` function.

aPath

A pointer to the path.

contourIndex

The index of the contour from which to get the point.

pointIndex

The index of the point to get.

thePoint

A pointer to a field that is to receive the point.

ptIsOnPath

A pointer to a field that is to receive a Boolean value that, if TRUE, specifies that the point is on the curve.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function lets programs get a single point from a path without walking the path data.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCodec.h`

CurveInsertPointIntoPath

Adds a new point to a path.

```
ComponentResult CurveInsertPointIntoPath (
   ComponentInstance effect,
   gxPoint *aPoint,
   Handle thePath,
   unsigned long contourIndex,
   unsigned long pointIndex,
   Boolean ptIsOnPath
);
```

Parameters

effect

The instance of the QuickTime vector codec component for the request. Your software obtains this reference when calling the Component Manager's `OpenComponent` or `OpenDefaultComponent` function.

aPoint

A pointer to the point to add to the path.

thePath

A handle to the path to which to add the point.

contourIndex

The index of the path contour to which to add the point.

pointIndex

The index of the point to add.

ptIsOnPath

If TRUE, specifies that the new point is to be on the path.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function is best for adding a single point to a path rather than large numbers of points.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qtvectors

qtvectors.win

Declared In

`ImageCodec.h`

CurveLengthToPoint

Obtains the point at a specified distance along a curve.

```
ComponentResult CurveLengthToPoint (
   ComponentInstance effect,
   gxPaths *target,
   long index,
   Fixed length,
   FixedPoint *location,
   FixedPoint *tangent
);
```

Parameters

effect

The instance of the QuickTime vector codec component for the request. Your software obtains this reference when calling the Component Manager's `OpenComponent` or `OpenDefaultComponent` function.

target

A pointer to the path.

index

The index of the path contour from which to get the point.

length

The distance along the curve at which to find the point.

location

A pointer to a field that is to receive the point.

tangent

A pointer to a field that is to receive a point that is tangent to the point at the specified distance.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function is useful for converting a value to a point, such as when creating an animation that follows a curve.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCodec.h`

CurveNewPath

Creates a new path.

```
ComponentResult CurveNewPath (
   ComponentInstance effect,
   Handle *pPath
);
```

Parameters

effect

The instance of the QuickTime vector codec component for the request. Your software obtains this reference when calling the Component Manager's `OpenComponent` or `OpenDefaultComponent` function.

pPath

A pointer to a handle that is to receive the new path.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The path created by this function contains one contour and no points. The caller must dispose of the handle when it is finished with it (with Macintosh, by calling `DisposeHandle`).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qtvectors

qtvectors.win

Declared In

`ImageCodec.h`

CurvePathPointToLength

Obtains the length of a path between specified starting and ending distances that is nearest a point.

```
ComponentResult CurvePathPointToLength (
   ComponentInstance ci,
   gxPaths *aPath,
   Fixed startDist,
   Fixed endDist,
   FixedPoint *thePoint,
   Fixed *pLength
);
```

Parameters

ci

The instance of the QuickTime vector codec component for the request. Your software obtains this reference when calling the Component Manager's `OpenComponent` or `OpenDefaultComponent` function.

aPath

A pointer to the path.

startDist
The distance along the path at which to start measuring the path's length.

endDist
The distance along the path at which to stop measuring the path's length.

thePoint
A pointer to a point; the function measures the path closest to this point.

pLength
A pointer to a field that is to receive the length of the specified part of the path.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
You can use this function to test if the user has clicked on the specified portion of the curve.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCodec.h`

CurveSetPathPoint

Changes the location of a point in a path.

```
ComponentResult CurveSetPathPoint (
   ComponentInstance effect,
   gxPaths *aPath,
   unsigned long contourIndex,
   unsigned long pointIndex,
   gxPoint *thePoint,
   Boolean ptIsOnPath
);
```

Parameters

effect
The instance of the QuickTime vector codec component for the request. Your software obtains this reference when calling the Component Manager's `OpenComponent` or `OpenDefaultComponent` function.

aPath
A pointer to the path.

contourIndex
The index of the path contour that contains the point to change.

pointIndex
The index of the point to change.

thePoint
A pointer to the new value for the point.

ptIsOnPath

If TRUE, specifies that the new point is to be on the path.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function edits an existing point location within a path. The function that you call to send the interpolated value to the receiving track is defined as a universal procedure in systems that support the Macintosh Code Fragment Manager (CFM) or is defined as a data procedure for non-CFM systems. With Macintosh, the `TweenerDataUPP` function pointer specifies the function the tween component calls with the value generated by the tween operation. A tween component calls this function from its implementation of the `TweenerDoTween` (page 549) function. You call this function by invoking the function specified in the tween record's `dataProc` field.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCodec.h`

DisposeImageCodecDrawBandCompleteUPP

Disposes of an ImageCodecDrawBandCompleteUPP pointer.

```
void DisposeImageCodecDrawBandCompleteUPP (
   ImageCodecDrawBandCompleteUPP userUPP
);
```

Parameters

userUPP

An `ImageCodecDrawBandCompleteUPP` pointer. See `Universal Procedure Pointers`.

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCodec.h`

DisposeImageCodecMPDrawBandUPP

Disposes of an ImageCodecMPDrawBandUPP pointer.

```
void DisposeImageCodecMPDrawBandUPP (
   ImageCodecMPDrawBandUPP userUPP
);
```

Parameters

userUPP

An `ImageCodecMPDrawBandUPP` pointer. See `Universal Procedure Pointers`.

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
ElectricImageComponent

ElectricImageComponent.win

OpenGLCompositorLab

SoftVideoOutputComponent

Declared In
`ImageCodec.h`

DisposeImageCodecTimeTriggerUPP

Disposes of an ImageCodecTimeTriggerUPP pointer.

```
void DisposeImageCodecTimeTriggerUPP (
   ImageCodecTimeTriggerUPP userUPP
);
```

Parameters

userUPP

An `ImageCodecTimeTriggerUPP` pointer. See `Universal Procedure Pointers`.

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCodec.h`

ImageCodecBandCompress

Asks your component to compress an image or a band of an image.

```
ComponentResult ImageCodecBandCompress (
   ComponentInstance ci,
   CodecCompressParams *params
);
```

Parameters

ci

An image decompressor component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

params

A pointer to a `CodecCompressParams` structure. The Image Compression Manager places the appropriate parameter information in that structure.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The image being compressed may be part of a sequence.

Special Considerations

Only compressors receive this request.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCodec.h`

ImageCodecBandDecompress

Asks your component to decompress a frame.

```
ComponentResult ImageCodecBandDecompress (
   ComponentInstance ci,
   CodecDecompressParams *params
);
```

Parameters

ci

An image decompressor component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

params

A pointer to a `CodecDecompressParams` structure. The Image Compression Manager places the appropriate parameter information in that structure.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion
For scheduled asynchronous decompression operations, the Image Compression Manager supplies a reference to an `ICMFrameTimeRecord` structure in this function's `CodecDecompressParams` structure parameter. The `ICMFrameTimeRecord` structure contains time information governing the scheduled decompression operation, including the time at which the frame must be displayed. For synchronous or immediate asynchronous decompress operations, the frame time is set to `NIL`.

When your component has finished the decompression operation, it must call the completion function. In the past, for asynchronous operations, your component called that function directly. For scheduled asynchronous decompression operations, your component should call `ICMDecompressComplete` (page 675).

If your component sets the `codecCanAsyncWhen` flag in predecompress but cannot support scheduled asynchronous decompression for a given frame, it must return an error code of `codecCantWhenErr`. If your component's queue is full, it should return an error code of `codecCantQueueErr`. Only decompressors receive these requests.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCodec.h`

ImageCodecBeginBand

Called before drawing a band or frame; it allows your image decompressor component to save information about a band before decompressing it.

```
ComponentResult ImageCodecBeginBand (
   ComponentInstance ci,
   CodecDecompressParams *params,
   ImageSubCodecDecompressRecord *drp,
   long flags
);
```

Parameters

ci

An image codec component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

params

A pointer to a `CodecDecompressParams` structure.

drp

A pointer to an `ImageSubCodecDecompressRecord` structure.

flags

Currently unused; set to 0.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Your image decompressor component receives the address of the destination pixel map in the `baseAddr` field of the `drp` parameter. This address includes an adjustment for the offset. Note that if the bit depth of the pixel map is less than 8, your image decompressor component must adjust for the bit offset.

The `codecData` field of the `drp` parameter contains a pointer to the compressed video data. The `userDecompressRecord` field of the `drp` parameter contains a pointer to storage for the decompression operation. The storage is allocated by the base image decompressor after it calls the `ImageCodecInitialize` (page 906) function. The size of the storage is determined by the `decompressRecordSize` field of the `ImageSubCodecDecompressCapabilities` structure that is returned by `ImageCodecInitialize` (page 906). Your image decompressor component should use this storage to store any additional information needed about the frame in order to decompress it.

Changes your image decompressor component makes to the `ImageSubCodecDecompressRecord` or `CodecDecompressParams` structures are preserved by the base image decompressor. For example, if your component does not need to decompress all of the data, it can change the pointer to the data to be decompressed that is stored in the `codecData` field of the `ImageSubCodecDecompressRecord` structure.

Special Considerations

Your component must implement this function. Also, the base image decompressor never calls `ImageCodecBeginBand` at interrupt time. If your component supports asynchronous scheduled decompression, it may receive more than one `ImageCodecBeginBand` call before receiving an `ImageCodecDrawBand` (page 880) call.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCodec.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.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`ImageCodec.h`

ImageCodecBusy

Lets your component report whether it is performing an asynchronous operation.

```
ComponentResult ImageCodecBusy (
   ComponentInstance ci,
   ImageSequence seq
);
```

Parameters

ci
: An image decompressor component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

seq
: The unique sequence identifier assigned by `CompressSequenceBegin` (page 609) or `DecompressSequenceBegin` (page 621).

Return Value
Your component should return a result code value of 1 if an asynchronous operation is in progress; it should return a result code value of 0 if the component is not performing an asynchronous operation. You can indicate an error by returning a negative result code. See `Error Codes`.

Special Considerations

Both compressors and decompressors may receive this request.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCodec.h`

ImageCodecCancelTrigger

Cancels an image codec's ImageCodecTimeTriggerProc callback.

```
ComponentResult ImageCodecCancelTrigger (
   ComponentInstance ci
);
```

Parameters

`ci`

An image codec component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

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` (page 34).

Availability
Available in Mac OS X v10.3 and later.

Declared In
`ImageCodec.h`

ImageCodecCreateStandardParameterDialog

Creates a parameters dialog box for a specified effect.

```
ComponentResult ImageCodecCreateStandardParameterDialog (
   ComponentInstance ci,
   QTAtomContainer parameterDescription,
   QTAtomContainer parameters,
   QTParameterDialogOptions dialogOptions,
   DialogPtr existingDialog,
   short existingUserItem,
   QTParameterDialog *createdDialog
);
```

Parameters

ci
An effects component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`. The dialog box that is created will allow the user to specify the parameters of this effect.

parameterDescription
The parameter description atom container for this effect. You can obtain a valid parameter description by calling `ImageCodecGetParameterList` (page 900). A parameter is optionally tweenable if defined as `kAtomInterpolateIsOptional` in its parameter description atom.

parameters
The atom container that will receive the user's chosen parameter values once the dialog has been dismissed.

dialogOptions
Controls how parameters containing tween data are presented in the created dialog box. If `dialogOptions` contains 0, two values are collected for each parameter that can be tweened, and the usual tweening operation will be performed for the duration of the effect being controlled. For other values,. See these constants:
`pdOptionsCollectOneValue`
`pdOptionsAllowOptionalInterpolations`

existingDialog
An existing dialog box that will have the controls from the standard parameters dialog box added to it. Set this parameter to `NIL` if you want this function to create a stand-alone dialog box.

existingUserItem
: The number of the user item in the existing dialog box that should be replaced with controls from the standard parameter dialog box. You should only pass a value to this parameter if the `existingDialog` parameter is not `NIL`.

createdDialog
: On return, a reference to the dialog created and displayed by the function. This reference is required by several other low-level effects functions. It will contain a valid dialog identifier even if you requested that the controls from the standard parameter dialog box be incorporated into an existing dialog box.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This is a low-level function that can be used to create a standard parameter dialog box for a specified effect, allowing the user to set the parameter values for the effect. You can optionally request that the controls from the dialog box be included within a dialog box of the calling application. The following sample code shows how to create a standard parameter dialog box and add effects controls:

```
// ImageCodecCreateStandardParameterDialog coding example
// See "Discovering QuickTime," page 303
pMovableModalDialog =GetNewDialog(kExtraDialogID, NIL, (WindowRef)-1);
if (pMovableModalDialog !=NIL) {
    ImageCodecCreateStandardParameterDialog(
        compInstance,
        qtacParameterDescription,
        qtacEffectSample,
        pdOptionsModelDialogBox,
        pMovableModalDialog,
        kExtraUserItemID,
        &lCreatedDialogID);
    ShowWindow(pMovableModalDialog);
    SelectWindow(pMovableModalDialog);
    SetPort(pMovableModalDialog);
}
```

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qtshoweffect

qtshoweffect.win

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.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`ImageCodec.h`

ImageCodecDismissStandardParameterDialog

Retrieves values from a standard parameter dialog box created by the low-level ImageCodecCreateStandardParameterDialog function, then closes the dialog box.

```
ComponentResult ImageCodecDismissStandardParameterDialog (
   ComponentInstance ci,
   QTParameterDialog createdDialog
);
```

Parameters

ci

An effect component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`. This must be the instance passed to `ImageCodecCreateStandardParameterDialog` (page 873) to create the dialog box.

createdDialog

A reference to the dialog box created by the call to `ImageCodecCreateStandardParameterDialog`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function should be called after the `ImageCodecIsStandardParameterDialogEvent` (page 907) function returns `codecParameterDialogConfirm` or `userCanceledErr`, which indicate that the user has dismissed the dialog box. The function dismisses the dialog box, deallocating any memory allocated during the call to `ImageCodecCreateStandardParameterDialog` (page 873).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qtshoweffect

qtshoweffect.win

Declared In
`ImageCodec.h`

ImageCodecDisposeImageGWorld

Disposes of an image graphics world associated with an image codec.

```
ComponentResult ImageCodecDisposeImageGWorld (
   ComponentInstance ci,
   GWorldPtr theGW
);
```

Parameters

ci
: An image codec component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

theGW
: A pointer to a `CGrafPort` structure.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCodec.h`

ImageCodecDisposeMemory

Disposes codec-allocated memory.

```
ComponentResult ImageCodecDisposeMemory (
   ComponentInstance ci,
   Ptr data
);
```

Parameters

ci
: An image compressor component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

data
: A pointer to the previously allocated memory block.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Your component receives the `ImageCodecDisposeMemory` request whenever an application calls `CDSequenceDisposeMemory` (page 590).

Special Considerations

When a codec instance is closed, it must ensure that all blocks allocated by that instance are disposed and call `ICMMemoryDisposedProc`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCodec.h`

ImageCodecDITLEvent

Lets an image codec component receive and process dialog events.

```
ComponentResult ImageCodecDITLEvent (
   ComponentInstance ci,
   DialogRef d,
   short itemOffset,
   const EventRecord *theEvent,
   short *itemHit,
   Boolean *handled
);
```

Parameters

`ci`

The component instance that identifies your connection to an image codec component.

`d`

A dialog reference identifying the settings dialog box.

`itemOffset`

The offset to your panel's first item in the dialog box.

`theEvent`

A pointer to an `EventRecord` structure. This structure contains information identifying the nature of the event.

`itemHit`

A pointer to a field that is to receive the item number in cases where your component handles the event. The number returned is an absolute, not a relative number, so it must be offset by the `itemOffset` parameter handled.

`handled`

A pointer to a Boolean value. Set this Boolean value to TRUE if you handle the event; set it to FALSE if you do not.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`ImageCodec.h`

ImageCodecDITLInstall

Installs added items in an image codec settings dialog box before the dialog box is displayed to the user.

```
ComponentResult ImageCodecDITLInstall (
   ComponentInstance ci,
   DialogRef d,
   short itemOffset
);
```

Parameters

`ci`
The component instance that identifies your connection to an image codec component.

`d`
A pointer to the dialog box to be installed.

`itemOffset`
The offset to your image codec's first dialog item.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`ImageCodec.h`

ImageCodecDITLItem

Receives and processes mouse clicks in the image codec settings dialog box.

```
ComponentResult ImageCodecDITLItem (
   ComponentInstance ci,
   DialogRef d,
   short itemOffset,
   short itemNum
);
```

Parameters

ci

The component instance that identifies your connection to an image codec component.

d

A dialog reference identifying the settings dialog box.

itemOffset

The offset to your panel's first item in the dialog box.

itemNum

The item number of the dialog item selected by the user. The sequence grabber provides an absolute item number. It is your responsibility to adjust this value to account for the offset to your panel's first item in the dialog box.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

An image codec component calls this function whenever the user clicks an item in the settings dialog box. Your component may then perform whatever processing is appropriate, depending upon the item number.

Version Notes

Introduced in QuickTime 6.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`ImageCodec.h`

ImageCodecDITLRemove

Removes a panel from the image codec settings dialog box.

```
ComponentResult ImageCodecDITLRemove (
   ComponentInstance ci,
   DialogRef d,
   short itemOffset
);
```

Parameters

ci

The component instance that identifies your connection to an image codec component.

d

A dialog pointer identifying the settings dialog box.

itemOffset

The offset to your panel's first item in the dialog box.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
An image codec component calls this function just before removing your items from the settings dialog box.

Version Notes
Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`ImageCodec.h`

ImageCodecDITLValidateInput

Validates the contents of the user dialog box for an image codec component.

```
ComponentResult ImageCodecDITLValidateInput (
   ComponentInstance ci,
   Boolean *ok
);
```

Parameters

`ci`

The component instance that identifies your connection to an image codec component.

`ok`

A pointer to a Boolean value. Set this value to TRUE if the settings are OK; otherwise, set it to FALSE.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
The image codec calls this function when the user clicks the OK button. If the user clicks the Cancel button, the image codec does not call this function. You indicate whether the settings are acceptable by setting the Boolean value pointed to by the `ok` parameter. If you set this value to FALSE, the sequence grabber component ignores the OK button in the dialog box.

Version Notes
Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`ImageCodec.h`

ImageCodecDrawBand

Decompresses a band or frame.

```
ComponentResult ImageCodecDrawBand (
   ComponentInstance ci,
   ImageSubCodecDecompressRecord *drp
);
```

Parameters

ci

An image decompressor component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

drp

A pointer to an `ImageSubCodecDecompressRecord` structure.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

When the base image decompressor calls your image decompressor component's `ImageCodecDrawBand` function, your component must perform the decompression specified by the fields of the `ImageSubCodecDecompressRecord` structure. The structure includes any changes your component made to it when performing the `ImageCodecBeginBand` (page 869) function. If your component supports asynchronous scheduled decompression, it may receive more than one `ImageCodecBeginBand` call before receiving an `ImageCodecDrawBand` call.

Special Considerations

Your component must implement this function. If the `ImageSubCodecDecompressRecord` structure specifies a progress function or data-loading function, the base image decompressor never calls this function at interrupt time. If not, the base image decompressor may call this function at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCodec.h`

ImageCodecDroppingFrame

Undocumented

```
ComponentResult ImageCodecDroppingFrame (
   ComponentInstance ci,
   const ImageSubCodecDecompressRecord *drp
);
```

Parameters

ci

An image codec component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

drp

A pointer to an `ImageSubCodecDecompressRecord` structure.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCodec.h`

ImageCodecEffectBegin

Undocumented

```
ComponentResult ImageCodecEffectBegin (
   ComponentInstance effect,
   CodecDecompressParams *p,
   EffectsFrameParamsPtr ePtr
);
```

Parameters

effect
An effect component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

p
A pointer to a `CodecDecompressParams` structure.

ePtr
A pointer to a `EffectsFrameParams` structure.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCodec.h`

ImageCodecEffectCancel

Undocumented

```
ComponentResult ImageCodecEffectCancel (
   ComponentInstance effect,
   EffectsFrameParamsPtr p
);
```

Parameters

`effect`

An effect component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

`p`

A pointer to a `EffectsFrameParams` structure.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCodec.h`

ImageCodecEffectConvertEffectSourceToFormat

Undocumented

```
ComponentResult ImageCodecEffectConvertEffectSourceToFormat (
   ComponentInstance effect,
   EffectSourcePtr sourceToConvert,
   ImageDescriptionHandle requestedDesc
);
```

Parameters

`effect`

An effect component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

`sourceToConvert`

Undocumented

`requestedDesc`

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Dimmer2Effect

Dimmer2Effect.win

GreyscaleEffectSample

Declared In
`ImageCodec.h`

ImageCodecEffectDisposeSMPTEFrame

Undocumented

```
ComponentResult ImageCodecEffectDisposeSMPTEFrame (
   ComponentInstance effect,
   SMPTEFrameReference frameRef
);
```

Parameters

`effect`
: An effect component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

`frameRef`
: *Undocumented*

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCodec.h`

ImageCodecEffectGetSpeed

Undocumented

```
ComponentResult ImageCodecEffectGetSpeed (
   ComponentInstance effect,
   QTAtomContainer parameters,
   Fixed *pFPS
);
```

Parameters

`effect`
: An effect component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

`parameters`
: *Undocumented*

`pFPS`
: *Undocumented*

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCodec.h`

ImageCodecEffectPrepareSMPTEFrame

Undocumented

```
ComponentResult ImageCodecEffectPrepareSMPTEFrame (
   ComponentInstance effect,
   PixMapPtr destPixMap,
   SMPTEFrameReference *returnValue
);
```

Parameters

effect
An effect component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

destPixMap
Undocumented

returnValue
Undocumented

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCodec.h`

ImageCodecEffectRenderFrame

Undocumented

```
ComponentResult ImageCodecEffectRenderFrame (
   ComponentInstance effect,
   EffectsFrameParamsPtr p
);
```

Parameters

effect

An effect component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

p

A pointer to an `EffectsFrameParams` structure.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCodec.h`

ImageCodecEffectRenderSMPTEFrame

Undocumented

```
ComponentResult ImageCodecEffectRenderSMPTEFrame (
   ComponentInstance effect,
   PixMapPtr destPixMap,
   SMPTEFrameReference frameRef,
   Fixed effectPercentageEven,
   Fixed effectPercentageOdd,
   Rect *pSourceRect,
   MatrixRecord *matrixP,
   SMPTEWipeType effectNumber,
   long xRepeat,
   long yRepeat,
   SMPTEFlags flags,
   Fixed penWidth,
   long strokeValue
);
```

Parameters

effect

An effect component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

destPixMap

Undocumented

frameRef

Undocumented

effectPercentageEven
: *Undocumented*

effectPercentageOdd
: *Undocumented*

pSourceRect
: *Undocumented*

pMatrix
: *Undocumented*

effectNumber
: *Undocumented*

xRepeat
: *Undocumented*

yRepeat
: *Undocumented*

flags
: *Undocumented*

penWidth
: *Undocumented*

strokeValue
: *Undocumented*

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCodec.h`

ImageCodecEffectSetup

Undocumented

```
ComponentResult ImageCodecEffectSetup (
   ComponentInstance effect,
   CodecDecompressParams *p
);
```

Parameters

effect
: An effect component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

p
: A pointer to a `CodecDecompressParams` structure.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCodec.h`

ImageCodecEncodeFrame

Presents the compressor with a frame to encode.

```
ComponentResult ImageCodecEncodeFrame (
   ComponentInstance ci,
   ICMCompressorSourceFrameRef sourceFrame,
   UInt32 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` (page 867).)

Availability
Available in Mac OS X v10.3 and later.

Declared In
`ImageCodec.h`

ImageCodecEndBand

Notifies your image decompressor component that decompression of a band has finished or that it was terminated by the Image Compression Manager.

```
ComponentResult ImageCodecEndBand (
   ComponentInstance ci,
   ImageSubCodecDecompressRecord *drp,
   OSErr result,
   long flags
);
```

Parameters

ci

An image decompressor component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

drp

A pointer to an `ImageSubCodecDecompressRecord` structure.

result

A result code.

flags

Currently unused; set to 0.

Return Value

Returns `noErr` if the band or frame was drawn successfully. If it is any other value, the band or frame was not drawn. See `Error Codes`.

Discussion

Your image decompressor component is not required to implement this function. After your image decompressor component handles a call to this function, it can perform any tasks that are required when decompression is finished, such as disposing of data structures that are no longer needed. Because this function can be called at interrupt time, your component cannot use it to dispose of data structures; this must occur after handling the function.

Special Considerations

The base image decompressor may call `ImageCodecEndBand` at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCodec.h`

ImageCodecExtractAndCombineFields

Performs field operations on video data.

```
ComponentResult ImageCodecExtractAndCombineFields (
   ComponentInstance ci,
   long fieldFlags,
   void *data1,
   long dataSize1,
   ImageDescriptionHandle desc1,
   void *data2,
   long dataSize2,
   ImageDescriptionHandle desc2,
   void *outputData,
   long *outDataSize,
   ImageDescriptionHandle descOut
);
```

Parameters

ci

An image decompressor component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

fieldFlags

Flags (see below) that specify the operation to be performed. A correctly formed request will specify two input fields, mapping one to the odd output field and the other to the even output field. See these constants:

`evenField1ToEvenFieldOut`

`evenField1ToOddFieldOut`

`oddField1ToEvenFieldOut`

`oddField1ToOddFieldOut`

`evenField2ToEvenFieldOut`

`evenField2ToOddFieldOut`

`oddField2ToEvenFieldOut`

`oddField2ToOddFieldOut`

data1

A pointer to a buffer containing the compressed image data for the first input field.

dataSize1

The size of the data1 buffer.

desc1

An `ImageDescription` structure describing the format and characteristics of the data in the data1 buffer.

data2

A pointer to a buffer containing the compressed image data for the second input field. Set to `NIL` if the requested operation uses only one input frame.

dataSize2

The size of the data2 buffer. Set to 0 if the requested operation uses only one input frame.

desc2

An `ImageDescription` structure describing the format and characteristics of the data in the data2 buffer. Set to `NIL` if the requested operation uses only one input frame.

outputData

A pointer to a buffer to receive the resulting frame.

outDataSize

On output this parameter returns the actual size of the new compressed image data.

descOut

The desired format of the resulting frames. Typically this is the same format specified by the desc1 and desc2 parameters.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function allows fields from two separate images, compressed in the same format, to be combined into a new compressed frame. Typically the operation results in an image of identical quality to the source images. Fields of a single image may also be duplicated or reversed by this function. The component selector for this function is:

```
kImageCodecExtractAndCombineFieldsSelect    =0x0015
```

Special Considerations

This codec routine implements the functionality of the `ImageFieldSequenceExtractCombine` (page 683) function. If your codec is capable of separately compressing both fields of a video frame, you should incorporate support for this function. Your codec must ensure that it understands the image formats specified by desc1 and desc2 parameters, as these may not be the same as the codec's native image format. The image format specified by the `descOut` parameter will be the same as the codec's native image format.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCodec.h`

ImageCodecFlush

Empties an image decompressor component's input queue of pending scheduled frames.

```
ComponentResult ImageCodecFlush (
   ComponentInstance ci
);
```

Parameters

ci

An image decompressor component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Your component receives the `ImageCodecFlush` function whenever the Image Compression Manager needs to cancel the display of all scheduled frames. Your decompressor should empty its queue of scheduled asynchronous decompression requests. For each request, your component must call

ICMDecompressComplete (page 675). Be sure to set the `err` parameter to -1, indicating that the request was canceled. Also, you must set both the `codecCompletionSource` and `codecCompletionDest` flags to 1.

Special Considerations

Your component's `ImageCodecFlush` function may be called at interrupt time. Only decompressor components that support scheduled asynchronous decompression receive this call.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCodec.h`

ImageCodecFlushFrame

Undocumented

```
ComponentResult ImageCodecFlushFrame (
   ComponentInstance ci,
   UInt32 flags
);
```

Parameters

ci

An image codec component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

flags

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCodec.h`

ImageCodecGetBaseMPWorkFunction

Gets an image codec's ComponentMPWorkFunctionProc callback.

```
ComponentResult ImageCodecGetBaseMPWorkFunction (
   ComponentInstance ci,
   ComponentMPWorkFunctionUPP *workFunction,
   void **refCon,
   ImageCodecMPDrawBandUPP drawProc,
   void *drawProcRefCon
);
```

Parameters

ci

An image codec component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

workFunction

On return, a pointer to a `ComponentMPWorkFunctionProc` callback.

refCon

On return, a handle to the reference constant that is passed to the `ComponentMPWorkFunctionProc` callback.

drawProc

An `ImageCodecMPDrawBandProc` callback.

drawProcRefCon

A pointer to a reference constant that is passed to your `ImageCodecMPDrawBandProc` callback. Use this parameter to point to a data structure containing any information your function needs.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

ElectricImageComponent

ElectricImageComponent.win

SoftVideoOutputComponent

Declared In

`ImageCodec.h`

ImageCodecGetCodecInfo

Notifies your codec whenever an application calls GetCodecInfo.

```
ComponentResult ImageCodecGetCodecInfo (
   ComponentInstance ci,
   CodecInfo *info
);
```

Parameters

ci

An image decompressor component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

info

A pointer to a `CodecInfo` structure to update. Your component should report its capabilities by formatting the structure in the location specified by this parameter.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Both compressors and decompressors may receive this request.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCodec.h`

ImageCodecGetCompressedImageSize

Notifies your codec whenever an application calls GetCompressedImageSize.

```
ComponentResult ImageCodecGetCompressedImageSize (
   ComponentInstance ci,
   ImageDescriptionHandle desc,
   Ptr data,
   long bufferSize,
   ICMDataProcRecordPtr dataProc,
   long *dataSize
);
```

Parameters

ci

An image decompressor component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

desc

A handle to the `ImageDescription` structure that defines the compressed image for the operation.

data

A pointer to the compressed image data.

bufferSize

The size of the buffer to be used by the data-loading function specified by the `dataProc` parameter. If the application did not specify a data-loading function this parameter is `NIL`.

dataProc

A pointer to an `ICMDataProcRecord` structure. If the data stream is not all in memory when the application calls `GetCompressedImageSize` (page 653), your component may call an application function that loads more compressed data.

dataSize

A pointer to a field that is to receive the size, in bytes, of the compressed image.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Special Considerations

Only decompressors receive this request.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCodec.h`

ImageCodecGetCompressionTime

Notifies your codec whenever an application calls GetCompressionTime.

```
ComponentResult ImageCodecGetCompressionTime (
   ComponentInstance ci,
   PixMapHandle src,
   const Rect *srcRect,
   short depth,
   CodecQ *spatialQuality,
   CodecQ *temporalQuality,
   unsigned long *time
);
```

Parameters

ci

An image decompressor component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

src

A handle to the source image. The source image is stored in a `PixMap` structure. Applications may use the time information you return for more than one image. Consequently, your compressor should not consider the contents of the image when determining the maximum compression time. Rather, you should consider only the quality level, pixel depth, and image size. This parameter may also be set to `NIL`. In this case the application has not supplied a source image; your component should use the `other` parameters to determine the characteristics of the image to be compressed.

srcRect

A pointer to a `Rect` structure that defines the portion of the source image to compress.

depth

The depth at which the image is to be compressed. Values of 1, 2, 4, 8, 16, 24, and 32 indicate the number of bits per pixel for color images. Values of 33, 34, 36, and 40 indicate 1-bit, 2-bit, 4-bit, and 8-bit grayscale, respectively, for grayscale images.

spatialQuality

A pointer to a field containing the desired compressed image quality (see below). The compressor sets this field to the closest actual quality that it can achieve. See these constants:

`codecMinQuality`

`codecLowQuality`

`codecNormalQuality`

`codecHighQuality`

`codecMaxQuality`

`codecLosslessQuality`

temporalQuality

A pointer to a field containing the desired sequence temporal quality (see below). The compressor sets this field to the closest actual quality that it can achieve.

time

A pointer to a field to receive the compression time, in milliseconds. If your component cannot determine the amount of time required to compress the image, set this field to 0. Check to see if the `value` of this field is `NIL` and, if so, do not write to location 0.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCodec.h`

ImageCodecGetDecompressLatency

Retrieves the video latency value from a specified video codec.

```
ComponentResult ImageCodecGetDecompressLatency (
   ComponentInstance ci,
   TimeRecord *latency
);
```

Parameters

ci

A video codec component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

latency

Pointer to a `TimeRecord` structure containing the latency time required for that codec.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCodec.h`

ImageCodecGetDITLForSize

Returns the size of various dialog item lists.

```
ComponentResult ImageCodecGetDITLForSize (
   ComponentInstance ci,
   Handle *ditl,
   Point *requestedSize
);
```

Parameters

`ci`
The component instance that identifies your connection to an image codec component.

`ditl`
A pointer to a handle provided by the sequence grabber component. Your panel component returns the dialog item list in this handle. Your component should resize this handle as appropriate. The sequence grabber component will dispose of this handle after retrieving the item list, so make sure that the item list is not stored in a resource.

`requestedSize`
The size of the panel in pixels, or constants (see below). Two special values, `kSGSmallestDITLSize` and `kSGLargestDITLSize`, request the smallest or largest size of the list. The sequence grabber will interpolate the panel elements between the two sizes if just the constants are returned. A codec must at a minimum support `kSGSmallestDITLSize` if it implements this call. See these constants:
`kSGSmallestDITLSize`
`kSGLargestDITLSize`

Return Value
The codec returns `badComponentSelector` for sizes it does not implement and `noErr` if there is no error. See `Error Codes`.

Discussion
This function allows an image codec to return dialog item lists of various size in pixels. Once you have created the area, you can use other image codec calls to handle the dialog items managed by your panel component.

Version Notes
Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`ImageCodec.h`

ImageCodecGetMaxCompressionSize

Notifies your codec whenever an application calls GetMaxCompressionSize.

```
ComponentResult ImageCodecGetMaxCompressionSize (
   ComponentInstance ci,
   PixMapHandle src,
   const Rect *srcRect,
   short depth,
   CodecQ quality,
   long *size
);
```

Parameters

ci

An image decompressor component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

src

A handle to the source image. The source image is stored in a `PixMap` structure. Applications use the size information you return to allocate buffers that may be used for more than one image. Consequently, your compressor should not consider the contents of the image when determining the maximum compressed size. Rather, you should consider only the quality level, pixel depth, and image size. This parameter may also be set to `NIL`. In this case the application has not supplied a source image; your component should use the `other` parameters to determine the characteristics of the image to be compressed.

srcRect

A pointer to a `Rect` structure that defines the portion of the source image to compress.

depth

The depth at which the image is to be compressed. Values of 1, 2, 4, 8, 16, 24, and 32 indicate the number of bits per pixel for color images. Values of 33, 34, 36, and 40 indicate 1-bit, 2-bit, 4-bit, and 8-bit grayscale, respectively, for grayscale images.

quality

The desired compressed image quality (see below). See these constants:

`codecMinQuality`

`codecLowQuality`

`codecNormalQuality`

`codecHighQuality`

`codecMaxQuality`

`codecLosslessQuality`

size

A pointer to a field to receive the maximum size, in bytes, of the compressed image.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The caller uses this function to determine the maximum size the data will become for a given parameter. Your component returns a long integer indicating the maximum number of bytes of compressed data that results from compressing the specified image.

Special Considerations

Only compressors receive this request.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCodec.h`

ImageCodecGetMaxCompressionSizeWithSources

Notifies your codec when an application calls GetCSequenceMaxCompressionSize.

```
ComponentResult ImageCodecGetMaxCompressionSizeWithSources (
   ComponentInstance ci,
   PixMapHandle src,
   const Rect *srcRect,
   short depth,
   CodecQ quality,
   CDSequenceDataSourcePtr sourceData,
   long *size
);
```

Parameters

`ci`
An image decompressor component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

`src`
A handle to the source image. The source image is stored in a `PixMap` structure. Applications use the size information you return to allocate buffers for more than one image. Consequently, your compressor should not consider the contents of the image when determining the maximum compressed size. Rather, you should consider only the quality level, pixel depth, and image size. This parameter may also be set to `NIL`. In this case the application has not supplied a source image; your component should use the `other` parameters to determine the characteristics of the image to be compressed.

`srcRect`
A pointer to a `Rect` structure that defines the portion of the source image to compress.

`depth`
The depth at which the image is to be compressed. Values of 1, 2, 4, 8, 16, 24, and 32 indicate the number of bits per pixel for color images. Values of 33, 34, 36, and 40 indicate 1-bit, 2-bit, 4-bit, and 8-bit grayscale, respectively, for grayscale images.

`quality`
The desired compression image quality.

`sourceData`
A pointer to a `CDSequenceDataSource` structure, which contains a linked list of all data sources. Because each data source contains a link to the next data source, a codec can access all data sources from this structure.

`size`
A pointer to a field to receive the maximum size, in bytes, of the compressed image.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
The caller uses this function to determine the maximum size the data will be compressed to for a given image and set of data sources.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCodec.h`

ImageCodecGetParameterList

Returns a parameter description atom container for a specified effect component instance.

```
ComponentResult ImageCodecGetParameterList (
   ComponentInstance ci,
   QTAtomContainer *parameterDescription
);
```

Parameters

`ci`

An effect component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

`parameterDescription`

The returned atom container for this component instance.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function returns the parameter description for the effect specified by the component instance `ci`, as a handle containing an `'atms'` resource of ID 1. The handle should be detached if it has been read in from a resource. Each parameter of the effect is described in the parameter description, with details of its name, type, legal values and hints about how a user interface to the parameter should be constructed.

Special Considerations

The calling application is responsible for disposing of the QT atom container returned in `parameterDescription`. The application should do this by calling `QTDisposeAtomContainer` (page 1427) once it has finished using the parameter description.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Fiendishthngs

qtshoweffect

qtshoweffect.win

Declared In

`ImageCodec.h`

ImageCodecGetParameterListHandle

Returns a handle to a Mac OS resource of type 'atms'.

```
ComponentResult ImageCodecGetParameterListHandle (
   ComponentInstance ci,
   Handle *parameterDescriptionHandle
);
```

Parameters

`ci`
: An effect component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

`parameterDescriptionHandle`
: A pointer to a handle to a Mac OS resource of type `'atms'`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The purpose of this function is to build a QT atom container in response to a call to `ImageCodecGetParameterList` (page 900). The handle should be detached if it has been read in from a resource. The caller is responsible for disposing of the handle.

Special Considerations

The implementation of this function in the Base Effect component reads in and detaches a Component Public Resource of type `'atms'` and ID 1.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCodec.h`

ImageCodecGetSettings

Returns the codec settings chosen by the user.

```
ComponentResult ImageCodecGetSettings (
   ComponentInstance ci,
   Handle settings
);
```

Parameters

`ci`
: An image compressor component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

settings
: A handle that the codec should resize and fill in with the current internal settings. If there are no current internal settings, resize it to 0. Don't dispose of this handle.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
`ImageCodecGetSettings` returns the codec's current internal settings. If there are no current settings or the settings are the same as the defaults, the codec can set the handle to `NIL`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Quartz Composer QCTV

Declared In
`ImageCodec.h`

ImageCodecGetSettingsAsText

Undocumented

```
ComponentResult ImageCodecGetSettingsAsText (
   ComponentInstance ci,
   Handle *text
);
```

Parameters

ci
: An image compressor component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

text
: *Undocumented*

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCodec.h`

ImageCodecGetSimilarity

Notifies your codec when an application calls GetSimilarity.

```
ComponentResult ImageCodecGetSimilarity (
   ComponentInstance ci,
   PixMapHandle src,
   const Rect *srcRect,
   ImageDescriptionHandle desc,
   Ptr data,
   Fixed *similarity
);
```

Parameters

ci

An image decompressor component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

src

A handle to the noncompressed image. The image is stored in a `PixMap` structure.

srcRect

A pointer to a `Rect` structure that defines the portion of the image to compare to the compressed image.

desc

A handle to the `ImageDescription` structure that defines the compressed image for the operation.

data

A pointer to the compressed image data.

similarity

A pointer to a field that is to receive the similarity value. Your component sets this field to reflect the relative similarity of the two images. Valid values range from 0 (key frame) to 1 (identical).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Only decompressors receive this request.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCodec.h`

ImageCodecGetSourceDataGammaLevel

Returns the native gamma of compressed data, if any.

```
ComponentResult ImageCodecGetSourceDataGammaLevel (
   ComponentInstance ci,
   Fixed *sourceDataGammaLevel
);
```

Parameters

ci

An image decompressor component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

sourceDataGammaLevel

On return, the native gamma level of the compressed data. If the value is 0, it is the default gamma level of the platform.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The ICM uses the information returned by this function to determine what gamma correction is necessary. For example, the Apple DV Codec returns 2.2.

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCodec.h`

ImageCodecHitTestData

Notifies your codec when the application calls PtInDSequenceData.

```
ComponentResult ImageCodecHitTestData (
   ComponentInstance ci,
   ImageDescriptionHandle desc,
   void *data,
   Size dataSize,
   Point where,
   Boolean *hit
);
```

Parameters

ci

An image decompressor component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

desc

An `ImageDescriptionHandle` for the image data pointed to by the `data` parameter.

data

Pointer to compressed data in the format specified by the `desc` parameter.

dataSize

`Size` of the compressed data referred to by the `data` parameter.

where
: A QuickDraw `Point` structure (0,0) based at the top-left corner of the image.

hit
: A pointer to a Boolean value. The value should be set to TRUE if the point specified by the `where` parameter is contained within the compressed image data specified by the `data` parameter, or FALSE if the specified point falls within a blank portion of the image.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
`ImageCodecHitTestData` allows the calling application to perform hit testing on compressed data.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCodec.h`

ImageCodecHitTestDataWithFlags

Undocumented

```
ComponentResult ImageCodecHitTestDataWithFlags (
   ComponentInstance ci,
   ImageDescriptionHandle desc,
   void *data,
   Size dataSize,
   Point where,
   long *hit,
   long hitFlags
);
```

Parameters

ci
: An image codec component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

desc
: A handle to an `ImageDescription` structure for the image data pointed to by the `data` parameter.

data
: Pointer to compressed data in the format specified by the `desc` parameter.

dataSize
: `Size` of the compressed data referred to by the `data` parameter.

where
: A QuickDraw `Point` structure (0,0) based at the top-left corner of the image.

hit
: A pointer to a Boolean. The Boolean should be set to TRUE if the point specified by the `where` parameter is contained within the compressed image data specified by the `data` parameter.

hitFlags

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCodec.h`

ImageCodecInitialize

Called before making any other all calls to your component.

```
ComponentResult ImageCodecInitialize (
   ComponentInstance ci,
   ImageSubCodecDecompressCapabilities *cap
);
```

Parameters

ci

An image decompressor component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

cap

On return, an `ImageSubCodecDecompressCapabilities` structure that contains the capabilities of the image decompressor component. This structure contains two fields. The `canAsync` field specifies whether your component can support asynchronous decompression operations. The `decompressRecordSize` field specifies the size of the decompression record structure for your component.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Your component must implement this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCodec.h`

ImageCodecIsImageDescriptionEquivalent

Compares image descriptions.

```
ComponentResult ImageCodecIsImageDescriptionEquivalent (
   ComponentInstance ci,
   ImageDescriptionHandle newDesc,
   Boolean *equivalent
);
```

Parameters

ci

An image compressor component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

newDesc

A handle to the `ImageDescription` structure that describes the compressed image.

equivalent

A pointer to a Boolean value. If the structure provided in the `newDesc` parameter is equivalent to the `ImageDescription` structure currently in use by the image sequence, this value is set to TRUE. If they are not equivalent, and therefore a new image sequence must be created to display an image using the new `ImageDescription` structure, this value is set to FALSE.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Your component receives this call whenever an application calls `CDSequenceEquivalentImageDescription` (page 591). Implementing this function can significantly improve playback of edited video sequences using your codec. For example, if two sequences are compressed at different quality levels and are edited together they will have different image descriptions because their quality values will be different. This will force QuickTime to use two separate decompressor instances to display the images. By implementing this function your decompressor can tell QuickTime that differences in quality levels don't require separate decompressors. This saves memory and time, thus improving performance.

Special Considerations

The current `ImageDescription` structure is not passed in this function because the Image Compression Manager assumes the codec has already made copies of all relevant data fields from the current `ImageDescription` structure during the `ImageCodecPreDecompress` (page 912) call.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCodec.h`

ImageCodecIsStandardParameterDialogEvent

Processes events related to a standard parameters dialog box created by ImageCodecCreateStandardParameterDialog.

```
ComponentResult ImageCodecIsStandardParameterDialogEvent (
   ComponentInstance ci,
   EventRecord *pEvent,
   QTParameterDialog createdDialog
);
```

Parameters

ci

An effect component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`. This must be the instance that was passed to `ImageCodecCreateStandardParameterDialog` (page 873) to create the dialog box.

pEvent

A pointer to an `EventRecord` structure.

createdDialog

A reference to the dialog box created by the call to `ImageCodecCreateStandardParameterDialog` (page 873).

Return Value

If the error code returned is `featureUnsupported`, your application should process the event in the normal way. If it is `noErr`, the event was processed. If this function returns any other value, an error occurred. See `Error Codes`.

Discussion

This function returns an error code that indicates whether the event pointed to by `pEvent` was processed or not. After you call `ImageCodecCreateStandardParameterDialog` (page 873) to create a standard parameter dialog box, you must pass every non-null event to this function. It processes events related to the standard parameter dialog box, passing other events to your application for processing.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qtshoweffect

qtshoweffect.win

Declared In

`ImageCodec.h`

ImageCodecMergeFloatingImageOntoWindow

Draws the current contents of a floating image.

```
ComponentResult ImageCodecMergeFloatingImageOntoWindow (
   ComponentInstance ci,
   UInt32 flags
);
```

Parameters

ci

The component instance that identifies your connection to an image codec component.

flags
: Currently not implemented.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Some hardware acceleration transfer codecs create a floating image in front of the window; when this is deactivated or hidden, whatever was previously drawn in that section of the window reappears. Such transfer codecs should implement this function, which draws the current contents of the floating image onto the window below, so that the floating image may be deactivated or hidden without the image changing.

Version Notes
Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`ImageCodec.h`

ImageCodecNewImageBufferMemory

Asks a codec to allocate memory for an offscreen buffer of non-RGB pixels.

```
ComponentResult ImageCodecNewImageBufferMemory (
   ComponentInstance ci,
   CodecDecompressParams *params,
   long flags,
   ICMMemoryDisposedUPP memoryGoneProc,
   void *refCon
);
```

Parameters

ci
: An image decompressor component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

params
: A pointer to a decompression parameters structure.

flags
: Currently, this parameter is always set to 0.

memoryGoneProc
: A pointer to an `ICMMemoryDisposedProc` callback that will be called before disposing of the memory allocated by the codec.

refCon
: A reference constant that is passed to your `ICMMemoryDisposedProc` callback. Use this parameter to point to a data structure containing any information your function needs.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This call is used to support a codec decompressing into a non-RGB buffer. The transfer codec is responsible for defining the offscreen buffer and transferring the image from the offscreen buffer to the destination. Your component receives this call whenever another codec has requested a non-RGB offscreen buffer of the type of your component's subtype.

Special Considerations

The Image Compression Manager does not currently track memory allocations. When a compressor or decompressor component instance is closed, it must ensure that all blocks allocated by that instance are disposed of and call the `ICMMemoryDisposedProc` callback.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCodec.h`

ImageCodecNewImageGWorld

Undocumented

```
ComponentResult ImageCodecNewImageGWorld (
   ComponentInstance ci,
   CodecDecompressParams *params,
   GWorldPtr *newGW,
   long flags
);
```

Parameters

`ci`
An image codec component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

`params`
A pointer to a `CodecDecompressParams` structure.

`newGW`
A pointer to a pointer to a `CGrafPort` structure.

`flags`
Undocumented

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCodec.h`

ImageCodecNewMemory

Requests codec-allocated memory.

```
ComponentResult ImageCodecNewMemory (
   ComponentInstance ci,
   Ptr *data,
   Size dataSize,
   long dataUse,
   ICMMemoryDisposedUPP memoryGoneProc,
   void *refCon
);
```

Parameters

ci

An image decompressor component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

data

Returns a pointer to the allocated memory.

dataSize

The desired size of the data buffer.

dataUse

A constant (see below) that indicates how the memory is to be used. For example, the memory may be used to store compressed data before it's displayed, mask plane data, or decompressed data. If there is no benefit to storing a particular kind of data in codec memory, the codec should refuse the request for the memory allocation. See these constants:

memoryGoneProc

A pointer to an `ICMMemoryDisposedProc` callback that will be called before disposing of the memory allocated by a codec.

refCon

A reference constant value that your codec must pass to the `memoryGoneProc` callback. Use this parameter to point to a data structure containing any information your callback needs.

Return Value

See `Error Codes`. Returns `noErr` if there is no error. If your codec does not currently have free memory for compression frame data, but will soon, you can return `codecNoMemoryPleaseWaitErr` to indicate this fact.

Discussion

Some hardware codecs may have on-board memory that can be used to store compressed and/or decompressed data. This function makes this memory available for use by clients of the codec. Some software codecs may be able to optimize their performance by having more control over memory allocation. This function makes such control available. Your component receives this call whenever an application calls `CDSequenceNewMemory` (page 597).

Special Considerations

The Image Compression Manager does not currently track memory allocations. When a compressor or decompressor component instance is closed, it must ensure that all blocks allocated by that instance are disposed and call the `ICMMemoryDisposedProc` callback.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCodec.h`

ImageCodecPreCompress

Notifies your component before compressing an image or a band of an image.

```
ComponentResult ImageCodecPreCompress (
   ComponentInstance ci,
   CodecCompressParams *params
);
```

Parameters

`ci`

An image decompressor component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

`params`

A pointer to a `CodecCompressParams` structure. The Image Compression Manager places the appropriate parameter information in that structure.

Return Value
See `Error Codes`. Your component should return a result code of `codecConditionErr` if it cannot field the compression request. Return `noErr` if there is no error.

Discussion
Your component receives this call before compressing an image or a band of an image. The Image Compression Manager also calls this function when processing a sequence. In that case, the Image Compression Manager calls this function whenever the parameters governing the sequence operation have changed substantially. Your component indicates whether it can perform the requested compression operation.

Special Considerations

Only compressors receive this call.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCodec.h`

ImageCodecPreDecompress

Notifies your component before decompressing an image or sequence of frames.

```
ComponentResult ImageCodecPreDecompress (
   ComponentInstance ci,
   CodecDecompressParams *params
);
```

Parameters

ci

An image decompressor component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

params

A pointer to a `CodecDecompressParams` structure. The Image Compression Manager places the appropriate parameter information in that structure.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

If your decompressor component supports scheduled asynchronous decompression operations, be sure to set the `codecCanAsyncWhen` flag to 1 in the `flags` field of your component's `CodecCapabilities` structure. If you set `codecCanAsyncWhen`, you must also set `codecCanAsync`. Codecs that support scheduled asynchronous decompression are strongly advised to also set the `codecCanShieldCursor` flag.

If your decompressor component uses a secondary hardware buffer for its images, be sure to set the `codecHasVolatileBuffer` flag to 1 in the `flags` field of your component's `CodecCapabilities` structure. If your decompressor component is used solely as a transfer codec and uses the `ImageCodecNewImageBufferMemory` (page 909) function to create an offscreen buffer that is really onscreen, your codec will need to set the `codecImageBufferIsOnScreen` flag to 1.

Special Considerations

Only decompressors receive this request.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCodec.h`

ImageCodecPreflight

Called before decompressing an image, in response to an ImageCodecPreDecompress call from the Image Compression Manager.

```
ComponentResult ImageCodecPreflight (
   ComponentInstance ci,
   CodecDecompressParams *params
);
```

Parameters

ci

An image decompressor component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

params
  A pointer to a `CodecDecompressParams` structure.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Your codec responds to this call by returning information about its capabilities in a `CodecCapabilities` structure. The Image Compression Manager creates the decompression parameters structure, and your image decompressor component is required only to provide values for the `wantedDestinationPixelSize` and `wantedDestinationPixelTypes` fields of the structure. Your image decompressor component can also modify other fields if necessary. For example, if it can scale images, it must set the `codecCapabilityCanScale` flag in the `capabilities` field of the structure.

Special Considerations
Your component must implement this function.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

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` (page 912).)

Availability

Available in Mac OS X v10.3 and later.

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`.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`ImageCodec.h`

ImageCodecQueueStarting

Called by the base image decompressor before decompressing the frames in the queue if your image decompressor component supports asynchronous scheduled decompression.

```
ComponentResult ImageCodecQueueStarting (
   ComponentInstance ci
);
```

Parameters

ci
: An image decompressor component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Your component is not required to implement this function. It can perform any tasks at this time, such as locking data structures.

Special Considerations

The base image decompressor never calls this function at interrupt time.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCodec.h`

ImageCodecQueueStopping

Notifies your component that the frames in the queue have been decompressed, if your image decompressor component supports asynchronous scheduled decompression.

```
ComponentResult ImageCodecQueueStopping (
   ComponentInstance ci
);
```

Parameters

ci

An image decompressor component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

After your image decompressor component handles a call to this function, it can perform any tasks that are required when decompression of the frames is finished, such as disposing of data structures that are no longer needed.

Special Considerations

Your component is not required to implement this function. This function is never called at interrupt time.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCodec.h`

ImageCodecRemoveFloatingImage

Hides an image codec's floating image without having to close the component.

```
ComponentResult ImageCodecRemoveFloatingImage (
   ComponentInstance ci,
   UInt32 flags
);
```

Parameters

ci

The component instance that identifies your connection to an image codec component.

flags

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Some hardware acceleration transfer codecs create a floating image in front of the window; when this is deactivated or hidden, whatever was previously drawn in that section of the window reappears. Such transfer codecs should implement this function, so the Image Compression Manager can ask it to hide the floating image without having to close the component. The floating image should be shown again on the next call to `ImageCodecDrawBand` (page 880).

Version Notes
Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`ImageCodec.h`

ImageCodecRequestGammaLevel

Asks an image codec to convert from source to destination gamma levels.

```
ComponentResult ImageCodecRequestGammaLevel (
   ComponentInstance ci,
   Fixed srcGammaLevel,
   Fixed dstGammaLevel,
   long *codecCanMatch
);
```

Parameters

`ci`

An image decompressor component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

`srcGammaLevel`

The gamma level to convert from.

`dstGammaLevel`

The gamma level to convert to.

`codecCanMatch`

Pointer to a value that indicates if the conversion from `srcGammaLevel` to `dstGammaLevel` is supported.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function tells the codec what the gamma of the source buffer and destination pixel map are so that the codec can try to convert between the two gammas when decompressing. Proper gamma conversion is accomplished by normalizing source data to black and white points to 0 to 1 and raising the result by the ratio of the `srcGammaLevel` divided by `dstGammaLevel`. The most accurate correction is done in RGB space, but a visual approximation can be done by raising the luma component alone.

Special Considerations

This function can be called several times as the ICM sets up a gamma conversion chain. The last value takes precedent for future scheduled frames. It may also be called while frames are already scheduled, indicating that conditions have changed. The new request is effective on frames that are scheduled after the call is made. Frames previously scheduled should continue to use the previously requested gamma conversion values.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCodec.h`

ImageCodecRequestSettings

Displays a dialog box containing codec-specific compression settings.

```
ComponentResult ImageCodecRequestSettings (
   ComponentInstance ci,
   Handle settings,
   Rect *rp,
   ModalFilterUPP filterProc
);
```

Parameters

ci
An image compressor component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

settings
A handle of data specific to the codec. If the handle is empty, the codec should use its default settings.

rp
A pointer to a `Rect` structure giving the coordinates of the standard compression dialog box in global screen coordinates. The codec can use this to position its dialog box in the same area of the screen.

filterProc
A pointer to a `ModalFilterProc` callback that the codec must either pass to the Mac OS `ModalDialog` function or call at the beginning of the codec dialog process. This callback gives the calling application and standard compression dialog box a chance to process update events.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
The `ImageCodecRequestSettings` function allows the display of a dialog box of additional compression settings specific to the codec. These settings are stored in a settings handle. The codec can store any data in any format it wants in the settings handle and resize it accordingly. It should store some type of tag or version information that it can use to verify that the data belongs to the codec. The codec should not dispose of the handle.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCodec.h`

ImageCodecScheduleFrame

Undocumented

```
ComponentResult ImageCodecScheduleFrame (
   ComponentInstance ci,
   const ImageSubCodecDecompressRecord *drp,
   ImageCodecTimeTriggerUPP triggerProc,
   void *triggerProcRefCon
);
```

Parameters

ci

An image codec component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

drp

A pointer to an `ImageSubCodecDecompressRecord` structure.

triggerProc

An `ImageCodecTimeTriggerProc` callback.

triggerProcRefCon

A reference constant that is passed to your `ImageCodecTimeTriggerProc` callback. Use this parameter to point to a data structure containing any information your function needs.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCodec.h`

ImageCodecSetSettings

Sets the settings of an optional image codec dialog box.

```
ComponentResult ImageCodecSetSettings (
   ComponentInstance ci,
   Handle settings
);
```

Parameters

ci

An image compressor component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

settings

A handle to internal settings originally returned by either `ImageCodecRequestSettings` (page 919) or `ImageCodecGetSettings` (page 901). The codec should set its internal settings to match those of the settings handle. Because the codec does not own the handle, it should not dispose of it and should copy only its contents, not the handle itself. If the settings handle passed is empty, the codec should sets its internal settings to a default state.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function allows a codec to return its private settings. Set the codec's internal settings to the state specified in the settings handle. The codec should always check the validity of the contents of the handle so that invalid settings are not used.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCodec.h`

ImageCodecSetTimeBase

Sets the time base for an image codec component.

```
ComponentResult ImageCodecSetTimeBase (
   ComponentInstance ci,
   void *base
);
```

Parameters

ci

An image codec component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

base

A pointer to the time base for this operation. Your application obtains this time base identifier from `NewTimeBase` (page 261).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCodec.h`

ImageCodecSetTimeCode

Sets the timecode for the next frame that is to be decompressed.

```
ComponentResult ImageCodecSetTimeCode (
   ComponentInstance ci,
   void *timeCodeFormat,
   void *timeCodeTime
);
```

Parameters

ci

An image decompressor component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

timeCodeFormat

A pointer to a `TimeCodeDef` structure. This structure contains the timecode definition information for the next frame to be decompressed.

timeCodeTime

A pointer to a `TimeCodeRecord` structure. This structure contains the time value for the next frame in the current sequence.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Your component receives this call whenever an application calls `SetDSequenceTimeCode` (page 728). That function allows an application to set the timecode for a frame that is to be decompressed. The timecode information you receive applies to the next frame to be decompressed and is provided to the decompressor by `ImageCodecBandDecompress` (page 868).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCodec.h`

ImageCodecSourceChanged

Notifies your codec that one of the data sources has changed when an application calls CDSequenceSetSourceData or CDSequenceChangedSourceData.

```
ComponentResult ImageCodecSourceChanged (
   ComponentInstance ci,
   UInt32 majorSourceChangeSeed,
   UInt32 minorSourceChangeSeed,
   CDSequenceDataSourcePtr sourceData,
   long *flagsOut
);
```

Parameters

ci

An image decompressor component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

majorSourceChangeSeed

An integer value that is incremented each time a data source is added or removed. This provides an easy way for a codec to know when it needs to redetermine which data source inputs are available.

minorSourceChangeSeed

An integer value that is incremented each time a data source is added or removed, or the data contained in any of the data sources changes. This provides a way for a codec to know if the data available to it has changed.

sourceData

A pointer to a `CDSequenceDataSource` structure. This structure contains a linked list of all data sources. Because each data source contains a link to the next data source, a codec can access all data sources from this structure.

flagsOut

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCodec.h`

ImageCodecStandardParameterDialogDoAction

Allows you to control the behavior of a standard parameter dialog box created by ImageCodecCreateStandardParameterDialog.

```
ComponentResult ImageCodecStandardParameterDialogDoAction (
   ComponentInstance ci,
   QTParameterDialog createdDialog,
   long action,
   void *params
);
```

Parameters

ci

An effect component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`. This must be the same instance as was passed to `ImageCodecCreateStandardParameterDialog` (page 873) to create the dialog box.

createdDialog

A reference to the dialog box created by the call to `ImageCodecCreateStandardParameterDialog`.

action

The action selector (see below), which determines which of the available actions you want the function to perform. See these constants:

- `pdActionConfirmDialog`
- `pdActionSetAppleMenu`
- `pdActionSetEditMenu`
- `pdActionGetDialogValues`
- `pdActionSetPreviewUserItem`
- `pdActionSetPreviewPicture`
- `pdActionSetColorPickerEventProc`
- `pdActionSetDialogTitle`
- `pdActionGetSubPanelMenu`
- `pdActionActivateSubPanel`
- `pdActionConductStopAlert`

params

The (optional) parameter to the `action`. The type passed in this parameter depends on the value of the `action` parameter.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function allows you to change the default behavior of the standard parameter dialog box, and provides a mechanism for your application to communicate with controls that were incorporated into an application dialog box. It also allows you to retrieve parameter values from the dialog box at any time. You specify which function will be performed by passing an action selector in the `action` parameter and, optionally, a single parameter in the `params` parameter. Some of the actions you can specify through this function are only appropriate if you have incorporated standard parameter dialog box controls within a dialog box created by your application.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qtshoweffect

qtshoweffect.win

Declared In

`ImageCodec.h`

ImageCodecTrimImage

Notifies your component whenever an application calls TrimImage.

```
ComponentResult ImageCodecTrimImage (
   ComponentInstance ci,
   ImageDescriptionHandle Desc,
   Ptr inData,
   long inBufferSize,
   ICMDataProcRecordPtr dataProc,
   Ptr outData,
   long outBufferSize,
   ICMFlushProcRecordPtr flushProc,
   Rect *trimRect,
   ICMProgressProcRecordPtr progressProc
);
```

Parameters

ci

An image decompressor component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

Desc

A handle to the `ImageDescription` structure that describes the compressed image. Your component updates this structure to refer to the resized image.

inData

A pointer to the compressed image data. If the entire compressed image cannot be stored at this location, the application may provide a data-loading function; see the description of the `dataProc` parameter to this function for details. This is a 32-bit clean address.

inBufferSize

The size of the buffer to be used by the data-loading function specified by the `dataProc` parameter. If the application did not specify a data-loading function, this parameter is `NIL`.

dataProc

A pointer to an `ICMDataProcRecord` structure. If the application did not provide a data-loading function, this parameter is `NIL`. In this case, the entire image must be in memory at the location specified by the `inData` parameter. If the data stream is not all in memory when the application calls `GetCompressedImageSize` (page 653), your component may call an application function that loads more compressed data.

outData

A pointer to a buffer to receive the trimmed image. If there is not sufficient memory to store the compressed image, the application may choose to write the compressed data to mass storage during the compression operation. The `flushProc` parameter identifies the data-unloading function. This is a 32-bit clean address.

outBufferSize
: The size of the buffer to be used by the data-unloading function specified by the `flushProc` parameter. If the application did not specify a data-unloading function, this parameter is `NIL`.

flushProc
: A pointer to an `ICMFlushProcRecord` structure. If the application did not provide a data-unloading function, this parameter is `NIL`. In this case, your component writes the entire compressed image into the memory location specified by the `outData` parameter. If there is not enough memory to store the compressed image, your component may call an application function that unloads some of the compressed data.

trimRect
: A pointer to a `Rect` structure that defines the desired image dimensions. Your component adjusts the structure's values so that they refer to the same rectangle in the resulting image. This is necessary whenever data is removed from the beginning of the image.

progressProc
: A pointer to an `ICMProgressProcRecord` structure. During the operation, your component should occasionally call an application function to report its progress. If the application did not provide a progress function, this parameter is `NIL`.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCodec.h`

ImageCodecValidateParameters

Validates effect parameters.

```
ComponentResult ImageCodecValidateParameters (
   ComponentInstance ci,
   QTAtomContainer parameters,
   QTParameterValidationOptions validationFlags,
   StringPtr errorString
);
```

Parameters

ci
: An image decompressor component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

parameters
: The atom container containing the `effect` parameters to be validated.

validationFlags
: Constants (see below) that control validation. See these constants:
  - `kParameterValidationNoFlags`
  - `kParameterValidationFinalValidation`

errorString

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCodec.h`

NewImageCodecDrawBandCompleteUPP

Allocates a Universal Procedure Pointer for an ImageCodecDrawBandCompleteProc callback.

```
ImageCodecDrawBandCompleteUPP NewImageCodecDrawBandCompleteUPP (
   ImageCodecDrawBandCompleteProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your application-defined function.

Return Value

A new UPP; see `Universal Procedure Pointers`.

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCodec.h`

NewImageCodecMPDrawBandUPP

Allocates a Universal Procedure Pointer for the ImageCodecMPDrawBandProc callback.

```
ImageCodecMPDrawBandUPP NewImageCodecMPDrawBandUPP (
   ImageCodecMPDrawBandProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your application-defined function.

Return Value

A new UPP; see `Universal Procedure Pointers`.

Discussion
This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes
Introduced in QuickTime 4.1. Replaces `NewImageCodecMPDrawBandProc`.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
ElectricImageComponent
ElectricImageComponent.win
SoftVideoOutputComponent

Declared In
`ImageCodec.h`

NewImageCodecTimeTriggerUPP

Allocates a Universal Procedure Pointer for the ImageCodecTimeTriggerProc callback.

```
ImageCodecTimeTriggerUPP NewImageCodecTimeTriggerUPP (
   ImageCodecTimeTriggerProcPtr userRoutine
);
```

Parameters

`userRoutine`
A pointer to your application-defined function.

Return Value
A new UPP; see `Universal Procedure Pointers`.

Discussion
This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes
Introduced in QuickTime 4.1. Replaces `NewImageCodecTimeTriggerProc`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCodec.h`

QTPhotoDefineHuffmanTable

Defines a Huffman table.

```
ComponentResult QTPhotoDefineHuffmanTable (
   ComponentInstance codec,
   short componentNumber,
   Boolean isDC,
   unsigned char *lengthCounts,
   unsigned char *values
);
```

Parameters

codec
Identifies your connection to the image compressor component.

componentNumber
Specifies a color component. If 0, the luminance Huffman table is set. If 1, the chrominance Huffman table is set.

isDC
If TRUE, the DC Huffman table is set. If FALSE, the AC Huffman table is set.

lengthCounts
A pointer to an array of 16 length counts.

values
A pointer to an array of Huffman values.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function lets you define a Huffman table to be used in future JPEG compression operations. Normally the JPEG image compressor components use the default Huffman tables as specified in sections K.3 through K.6 of the JPEG specification. You can use this function to override the default tables.

Special Considerations
This call is supported only by the Photo JPEG and Motion JPEG compressors. Only advanced programmers will need to use this function.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCodec.h`

QTPhotoDefineQuantizationTable

Specifies a custom quantization table.

```
ComponentResult QTPhotoDefineQuantizationTable (
   ComponentInstance codec,
   short componentNumber,
   unsigned char *table
);
```

Parameters

codec

Identifies your connection to the image compressor component.

componentNumber

If 0, the luminance quantization table is set. If 1, the chrominance quantization table is set.

table

A pointer to an array of 64 quantization values.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

By default, the JPEG compressors select quantization tables based on quality settings. This function lets you override these tables with tables of your own choice.

Special Considerations

This call is only supported by the Photo JPEG and Motion JPEG compressors. Only advanced programmers will need to use this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCodec.h`

QTPhotoSetRestartInterval

Specifies the restart interval to use in future JPEG compression operations.

```
ComponentResult QTPhotoSetRestartInterval (
   ComponentInstance codec,
   unsigned short restartInterval
);
```

Parameters

codec

Identifies your connection to the image compressor component.

restartInterval

The new restart interval. Pass 0 to tell the compressor not to insert restart markers in the data stream.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

By default, the JPEG compressor components do not insert restart markers in the compressed data stream unless the "optimize for streaming" setting is selected.

Special Considerations

This call is supported only by the Photo JPEG and Motion JPEG compressors. Only advanced programmers will need to use this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCodec.h`

QTPhotoSetSampling

Specifies the chrominance downsampling ratio to use in future JPEG compression operations.

```
ComponentResult QTPhotoSetSampling (
   ComponentInstance codec,
   short yH,
   short yV,
   short cbH,
   short cbV,
   short crH,
   short crV
);
```

Parameters

codec

Identifies your connection to the image compressor component.

yH

The number of horizontal luminance blocks to put in each macroblock.

yV

The number of vertical luminance blocks to put in each macroblock.

cbH

The number of horizontal chroma blue blocks to put in each macroblock.

cbV

The number of vertical chroma blue blocks to put in each macroblock.

crH

The number of horizontal chroma red blocks to put in each macroblock.

crV

The number of vertical chroma red blocks to put in each macroblock.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

By default, the Photo JPEG compressor uses 4:1:1 chroma downsampling and the Motion JPEG compressors use 4:2:2 chroma downsampling for most quality settings. For `codecLosslessQuality`, both compressors disable chroma downsampling. Currently the only supported downsampling ratios are none (pass 1,1,1,1,1,1), 4:2:2 (pass 2,1,1,1,1,1) and 4:1:1 (pass 2,2,1,1,1,1).

Special Considerations

This call is supported only by the Photo JPEG and Motion JPEG compressors. Only advanced programmers will need to use this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCodec.h`

Callbacks

ComponentMPWorkFunctionProc

Undocumented

```
typedef ComponentResult (*ComponentMPWorkFunctionProcPtr) (void *globalRefCon,
ComponentMPWorkFunctionHeaderRecordPtr header);
```

If you name your function `MyComponentMPWorkFunctionProc`, you would declare it this way:

```
ComponentResult MyComponentMPWorkFunctionProc (
    void                                     *globalRefCon,
    ComponentMPWorkFunctionHeaderRecordPtr    header );
```

Parameters

globalRefCon

Undocumented

header

Pointer to a `ComponentMPWorkFunctionHeaderRecord` structure.

Return Value

See `Error Codes`. Your callback should return `noErr` if there is no error.

Declared In

`ImageCodec.h`

ImageCodecMPDrawBandProc

Undocumented

```
typedef ComponentResult (*ImageCodecMPDrawBandProcPtr) (void *refcon,
ImageSubCodecDecompressRecord *drp);
```

If you name your function `MyImageCodecMPDrawBandProc`, you would declare it this way:

```
ComponentResult MyImageCodecMPDrawBandProc (
    void                            *refcon,
    ImageSubCodecDecompressRecord    *drp );
```

Parameters

refcon

Pointer to a reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

drp

Pointer to an `ImageSubCodecDecompressRecord` structure.

Return Value

See `Error Codes`. Your callback should return `noErr` if there is no error.

Declared In

`ImageCodec.h`

ImageCodecTimeTriggerProc

Undocumented

```
typedef void (*ImageCodecTimeTriggerProcPtr) (void *refcon);
```

If you name your function `MyImageCodecTimeTriggerProc`, you would declare it this way:

```
void MyImageCodecTimeTriggerProc (
    void    *refcon );
```

Parameters

refcon

Pointer to a reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

Declared In

`ImageCodec.h`

Data Types

CDSequenceDataSource

Contains a linked list of all data sources for a decompression sequence.

```
struct CDSequenceDataSource {
     long                        recordSize;
     void *                      next;
     ImageSequence               seqID;
     ImageSequenceDataSource     sourceID;
     OSType                      sourceType;
     long                        sourceInputNumber;
     void *                      dataPtr;
     Handle                      dataDescription;
     long                        changeSeed;
     ICMConvertDataFormatUPP     transferProc;
     void *                      transferRefcon;
     long                        dataSize;
     QHdrPtr                     dataQueue;
     void *                      originalDataPtr;
     long                        originalDataSize;
     Handle                      originalDataDescription;
     long                        originalDataDescriptionSeed;
 };
```

Fields

`recordSize`

Discussion

The size of this structure.

`next`

Discussion

A pointer to the next source entry. If it is `NIL`, there are no more entries.

`seqID`

Discussion

The image sequence that this source is associated with.

`sourceID`

Discussion

The source reference identifying this source.

`sourceType`

Discussion

A four-character code describing how the input will be used. This value is passed to this parameter by `CDSequenceNewDataSource` (page 595) when the source is created.

`sourceInputNumber`

Discussion

A value is passed to this parameter by `CDSequenceNewDataSource` when the source is created.

`dataPtr`

Discussion

A pointer to the actual source data.

`dataDescription`

Discussion

A handle to a data structure describing the data format. This is often a handle to an `ImageDescription` structure.

`changeSeed`

Discussion
An integer that is incremented each time the `dataPtr` field changes or that data that the `dataPtr` field points to changes. By remembering the `value` of this field and comparing to the value the next time the decompressor or compressor component is called, the component can determine if new data is present.

`transferProc`

Discussion
Reserved.

`transferRefcon`

Discussion
Reserved.

`dataSize`

Discussion
The size of the data pointed to by the `dataPtr` field.

`dataQueue`

Discussion
A pointer to a `QHdr` structure that contains a queue of `CDSequenceDataSourceQueueEntry` structures.

`originalDataPtr`

Discussion
The original value of `dataPtr`.

`originalDataSize`

Discussion
The original value of `dataSize`.

`originalDataDescription`

Discussion
The original value of `dataDescription`.

`originalDataDescriptionSeed`

Discussion
The original value of `changeSeed`.

Discussion
Because each data source is associated with a link to the next data source, a codec can access all data sources using this structure.

Version Notes
Fields from `dataQueue` onward were introduced in QuickTime 3.

Related Functions
`ImageCodecGetMaxCompressionSizeWithSources` (page 899)
`ImageCodecSourceChanged` (page 922)

Declared In
`ImageCodec.h`

CDSequenceDataSourcePtr

Represents a type used by the Image Codec API.

```
typedef CDSequenceDataSource * CDSequenceDataSourcePtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCodec.h`

CodecCompressParams

Contains parameters that govern a compression operation.

```
struct CodecCompressParams {
     ImageSequence              sequenceID;
     ImageDescriptionHandle     imageDescription;
     Ptr                        data;
     long                       bufferSize;
     long                       frameNumber;
     long                       startLine;
     long                       stopLine;
     long                       conditionFlags;
     CodecFlags                 callerFlags;
     CodecCapabilities *        capabilities;
     ICMProgressProcRecord      progressProcRecord;
     ICMCompletionProcRecord    completionProcRecord;
     ICMFlushProcRecord         flushProcRecord;
     PixMap                     srcPixMap;
     PixMap                     prevPixMap;
     CodecQ                     spatialQuality;
     CodecQ                     temporalQuality;
     Fixed                      similarity;
     DataRateParamsPtr          dataRateParams;
     long                       reserved;
     UInt16                     majorSourceChangeSeed;
     UInt16                     minorSourceChangeSeed;
     CDSequenceDataSourcePtr    sourceData;
     long                       preferredPacketSizeInBytes;
     long                       requestedBufferWidth;
     long                       requestedBufferHeight;
     OSType                     wantedSourcePixelType;
     long                       compressedDataSize;
     UInt32                     taskWeight;
     OSType                     taskName;
 };
```

Fields
`sequenceID`

Discussion
Contains a unique sequence identifier. If the image to be compressed is part of a sequence, this field contains the sequence identifier that was assigned by `CompressSequenceBegin` (page 609). If the image is not part of a sequence, this field is set to 0.

`imageDescription`

Discussion
Contains a handle to the image description structure that describes the image to be compressed.

`data`

Discussion
Points to a location to receive the compressed image data. This is a 32-bit clean address. If there is not sufficient memory to store the compressed image, the application may choose to write the compressed data to mass storage during the compression operation. The `flushProcRecord` field identifies the data-unloading function that the application provides for this purpose. This field is used only by `ImageCodecBandCompress` (page 867).

`bufferSize`

Discussion
Contains the size of the buffer specified by the `data` field. Your component sets the value of the `bufferSize` field to the number of bytes of compressed data written into the buffer. Your component should not return more data than the buffer can hold; it should return a nonzero result code instead. This field is used only by `ImageCodecBandCompress` (page 867).

`frameNumber`

Discussion
Contains a frame identifier. Indicates the relative frame number within the sequence. The Image Compression Manager increments this value for each frame in the sequence. This field is used only by `ImageCodecBandCompress` (page 867).

`startLine`

Discussion
Contains the starting line for the band. This field indicates the starting line number for the band to be compressed. The line number refers to the pixel row in the image, starting from the top of the image. The first row is row number 0. This field is used only by `ImageCodecBandCompress` (page 867).

`stopLine`

Discussion
Contains the ending line for the band. This field indicates the ending line number for the band to be compressed. The line number refers to the pixel row in the image, starting from the top of the image. The first row in the image is row number 0. The image band includes the row specified by this field. So, to define a band that contains one row of pixels at the top of an image, you set the `startLine` field to 0 and the `stopLine` field to 1.

`conditionFlags`

Discussion
Contains flags (see below) that identify the condition under which your component has been called. This field is used only by `ImageCodecBandCompress` (page 867). In addition, these fields contain information about actions taken by your component. See these constants:

`codecConditionFirstBand`

`codecConditionLastBand`

`codecConditionCodecChangedMask`

`callerFlags`

Discussion
Flags that provide further control information. This field is used only by `ImageCodecBandCompress` (page 867). See these constants:

- `codecFlagUpdatePrevious`
- `codecFlagWasCompressed`
- `codecFlagUpdatePreviousComp`
- `codecFlagLiveGrab`

`capabilities`

Discussion
Points to a compressor capability structure. The Image Compression Manager uses this field to determine the capabilities of your compressor component. This field is used only by `ImageCodecPreCompress` (page 912).

`progressProcRecord`

Discussion
Contains an `ICMProgressProcRecord` structure. During the compression operation, your compressor may occasionally call a function that the application provides in order to report your progress. This field contains a structure that identifies the progress function. If the `progressProc` field in this structure is set to `NIL`, the application has not supplied a progress function. This field is used only by `ImageCodecBandCompress` (page 867).

`completionProcRecord`

Discussion
Contains an `ICMCompletionProcRecord` structure. This structure governs whether you perform the compression asynchronously. If the `completionProc` field in this structure is set to `NIL`, perform the compression synchronously. If this field is not `NIL`, it specifies an application completion function. Perform the compression asynchronously and call that completion function when your component is finished. If the `completionProc` field in this structure has a value of -1, perform the operation asynchronously but do not call the application's completion function. This field is used only by `ImageCodecBandCompress` (page 867).

`flushProcRecord`

Discussion
Contains an `ICMFlushProcRecord` structure. If there is not enough memory to store the compressed image, the application may provide a function that unloads some of the compressed data. This field contains a structure that identifies that data-unloading function. If the application did not provide a data-unloading function, the `flushProc` field in this structure is set to `NIL`. In this case, your component writes the entire compressed image into the memory location specified by the `data` field. The data-unloading function structure is used only by `ImageCodecBandCompress` (page 867).

`srcPixMap`

Discussion
Points to the image to be compressed. The image must be stored in a pixel map structure. The contents of this pixel map differ from a standard pixel map in two ways. First, the `rowBytes` field is a full 16-bit value; the high-order bit is not necessarily set to 1. Second, the `baseAddr` field must contain a 32-bit clean address. This field is used only by `ImageCodecBandCompress` (page 867).

`prevPixMap`

Discussion
Points to a pixel map containing the previous image. If the image to be compressed is part of a sequence that is being temporally compressed, this field defines the previous image for temporal compression. Your component should then use this previous image as the basis of comparison for the image to be compressed. If the `temporalQuality` field is set to 0, do not perform temporal compression. If the `codecFlagUpdatePrevious` flag or the `codecFlagUpdatePreviousComp` flag in the `flags` field is set to 1, update the previous image at the end of the compression operation. The contents of this pixel map differ from a standard pixel map in two ways. First, the `rowBytes` field is a full 16-bit value; the high-order bit is not necessarily set to 1. Second, the `baseAddr` field must contain a 32-bit clean address. This field is used only by `ImageCodecBandCompress` (page 867).

`spatialQuality`

Discussion
Specifies the desired compressed image quality. This field is used only by `ImageCodecBandCompress` (page 867).

`temporalQuality`

Discussion
Specifies the desired sequence temporal quality. This field governs the level of compression the application desires with respect to information in successive frames in the sequence. If this field is set to 0, do not perform temporal compression on this frame. This field is used only by `ImageCodecBandCompress` (page 867).

`similarity`

Discussion
Indicates the relative similarity between the frame just compressed and the previous frame when performing temporal compression. Fixed-point value, ranges from 0 (0x00000000), indicating a key frame, to 255 (0x00FF0000), indicating an identical frame that can be discarded without damage. If bad video would result from discarding a frame, the compressor should limit similarity to 254 (0x00FE0000). The Image Compression Manager may request a compressor to recompress a frame as a key frame if its similarity to its predecessor is very low (a value of 1 or 2, for example). The Image Compression Manager will not do this if the codecFlagLiveGrab flag is set, or if an asynchronous completion proc is supplied. This field is used only by `ImageCodecBandCompress` (page 867).

`dataRateParams`

Discussion
Points to the parameters used when performing data rate constraint.

`reserved`

Discussion
Reserved.

`majorSourceChangeSeed`

Discussion
Contains an integer value that is incremented each time a data source is added or removed. This provides a fast way for a codec to know when it needs to redetermine which data source inputs are available.

`minorSourceChangeSeed`

Discussion
Contains an integer value that is incremented each time a data source is added or removed, or the data contained in any of the data sources changes. This provides a way for a codec to know if the data available to it has changed.

`sourceData`

Discussion
Contains a pointer to a `CDSequenceDataSource` structure. This structure contains a linked list of all data sources. Because each data source contains a link to the next data source, a codec can access all data sources from this field.

`preferredPacketSizeInBytes`

Discussion
Specifies the preferred packet size for data.

`requestedBufferWidth`

Discussion
Specifies the the width of the image buffer to use, in pixels. For this value to be used, the `codecWantsSpecialScaling` flag in the `CodecCapabilities` structure must be set.

`requestedBufferHeight`

Discussion
Specifies the the height of the image buffer to use, in pixels. For this value to be used, the `codecWantsSpecialScaling` flag in the `CodecCapabilities` structure must be set.

`wantedSourcePixelType`

Discussion
Undocumented

`compressedDataSize`

Discussion
The size of the compressed image, in bytes. If this field is nonzero, it overrides the `dataSize` field of the `ImageDescription` structure. This provides a safer way for asynchronous compressors to return the size of the compressed frame data, because the `dataSize` field of `ImageDescription` may be referenced by an unlocked handle.

`taskWeight`

Discussion
The preferred weight for multiprocessing tasks implementing this operation. You should assign a value by means of the Mac OS function `MPSetTaskWeight`.

`taskName`

Discussion
The preferred type for multiprocessing tasks implementing this operation. You should assign a value by means of the Mac OS function `MPSetTaskType`.

Discussion
Compressor components accept the parameters that govern a compression operation in the form of the `CodecCompressParams` structure. This structure is used by `ImageCodecBandCompress` (page 867) and `ImageCodecPreCompress` (page 912).

Version Notes
Some of the fields in `CodecCompressParams` were added for various versions of QuickTime starting with version 2.1. See comments in the C interface file for details.

Related Functions
`ImageCodecBandCompress` (page 867)
`ImageCodecPreCompress` (page 912)

Declared In
`ImageCodec.h`

CodecDecompressParams

The basic parameter block that is passed to a decompressor.

```
struct CodecDecompressParams {
     ImageSequence               sequenceID;
     ImageDescriptionHandle      imageDescription;
     Ptr                         data;
     long                        bufferSize;
     long                        frameNumber;
     long                        startLine;
     long                        stopLine;
     long                        conditionFlags;
     CodecFlags                  callerFlags;
     CodecCapabilities *         capabilities;
     ICMProgressProcRecord       progressProcRecord;
     ICMCompletionProcRecord     completionProcRecord;
     ICMDataProcRecord           dataProcRecord;
     CGrafPtr                    port;
     PixMap                      dstPixMap;
     BitMapPtr                   maskBits;
     PixMapPtr                   mattePixMap;
     Rect                        srcRect;
     MatrixRecord *              matrix;
     CodecQ                      accuracy;
     short                       transferMode;
     ICMFrameTimePtr             frameTime;
     long                        reserved[1];
     SInt8                       matrixFlags;
     SInt8                       matrixType;
     Rect                        dstRect;
     UInt16                      majorSourceChangeSeed;
     UInt16                      minorSourceChangeSeed;
     CDSequenceDataSourcePtr     sourceData;
     RgnHandle                   maskRegion;
     OSType **                   wantedDestinationPixelTypes;
     long                        screenFloodMethod;
     long                        screenFloodValue;
     short                       preferredOffscreenPixelSize;
     ICMFrameTimeInfoPtr         syncFrameTime;
     Boolean                     needUpdateOnTimeChange;
     Boolean                     enableBlackLining;
     Boolean                     needUpdateOnSourceChange;
     Boolean                     pad;
     long                        unused;
     CGrafPtr                    finalDestinationPort;
     long                        requestedBufferWidth;
     long                        requestedBufferHeight;
     Rect                        displayableAreaOfRequestedBuffer;
     Boolean                     requestedSingleField;
     Boolean                     needUpdateOnNextIdle;
     Boolean                     pad2[2];
     fixed                       bufferGammaLevel;
     UInt32                      taskWeight;
     OSType                      taskName;
 };
```

Fields

`sequenceID`

Discussion
Contains the unique sequence identifier. If the image to be decompressed is part of a sequence, this field contains the sequence identifier that was assigned by `DecompressSequenceBegin` (page 621). If the image is not part of a sequence, this field is set to 0.

`imageDescription`

Discussion
Contains a handle to the `ImageDescription` that describes the image to be decompressed.

`data`

Discussion
Points to the compressed image data. This must be a 32-bit clean address. The `bufferSize` field indicates the size of this data buffer. If the entire compressed image does not fit in memory, the application should provide a data-loading function, identified by the `dataProc` field of the data-loading function structure stored in the `dataProcRecord` field. This field is used only by `ImageCodecBandDecompress` (page 868).

`bufferSize`

Discussion
Specifies the size of the image data buffer. This field is used only by `ImageCodecBandDecompress` (page 868).

`frameNumber`

Discussion
Contains a frame identifier. Indicates the relative frame number within the sequence. The Image Compression Manager increments this value for each frame in the sequence. This field is used only by `ImageCodecBandDecompress` (page 868).

`startLine`

Discussion
Specifies the starting line for the band. The line number refers to the pixel row in the image, starting from the top of the image. The first row in the image is row number 0. This field is used only by `ImageCodecBandDecompress` (page 868).

`stopLine`

Discussion
Specifies the ending line for the band. The line number refers to the pixel row in the image, starting from the top of the image. The first row is row number 0. The image band includes the row specified by this field. So, to define a band that contains one row of pixels at the top of an image, you set the `startLine` field to 0 and the `stopLine` field to 1. This field is used only by `ImageCodecBandDecompress` (page 868).

`conditionFlags`

Discussion

Contains flags (see below) that identify the condition under which your component has been called (in order to save the component some work). The flags in this field are passed to the component by `ImageCodecBandCompress` (page 867) and `ImageCodecPreDecompress` (page 912) when conditions change, to save it some work. In addition, these fields contain information about actions taken by your component. See these constants:

- `codecConditionFirstBand`
- `codecConditionLastBand`
- `codecConditionFirstFrame`
- `codecConditionNewDepth`
- `codecConditionNewTransform`
- `codecConditionNewSrcRect`
- `codecConditionNewMatte`
- `codecConditionNewTransferMode`
- `codecConditionNewClut`
- `codecConditionNewAccuracy`
- `codecConditionNewDestination`
- `codecConditionCodecChangedMask`
- `codecConditionFirstScreen`
- `codecConditionDoCursor`
- `codecConditionCatchUpDiff`
- `codecConditionMaskMayBeChanged`
- `codecConditionToBuffer`

`callerFlags`

Discussion

Contains flags (see below) that provide further control information. This field is used only by `ImageCodecBandCompress` (page 867). See these constants:

- `codecFlagUpdatePrevious`
- `codecFlagWasCompressed`
- `codecFlagUpdatePreviousComp`
- `codecFlagLiveGrab`

`capabilities`

Discussion

Points to a `CodecCapabilities` structure. The Image Compression Manager uses this parameter to determine the capabilities of your decompressor component. This field is used only by `ImageCodecPreDecompress` (page 912).

`progressProcRecord`

Discussion

Contains a `ICMProgressProcRecord` structure. During the decompression operation, your decompressor may occasionally call a function that the application provides in order to report your progress. This field contains a structure that identifies the progress function. If the `progressProc` field of this structure is set to `NIL`, the application did not provide a progress function. This field is used only by `ImageCodecBandDecompress` (page 868).

`completionProcRecord`

Discussion
Contains an `ICMCompletionProcRecord` structure. This field governs whether you perform the decompression asynchronously. If the `completionProc` field in this structure is set to `NIL`, perform the decompression synchronously. If this field is not `NIL`, it specifies an application completion function. Perform the decompression asynchronously and call that completion function when your component is finished. If this field has a value of -1, perform the operation asynchronously but do not call the application's completion function. This field is used only by `ImageCodecBandDecompress` (page 868).

`dataProcRecord`

Discussion
Contains an `ICMDataProcRecord` structure. If the data stream is not all in memory, your component may call an application function that loads more compressed data. This field contains a structure that identifies that data-loading function. If the application did not provide a data-loading function, the `dataProc` field in this structure is set to `NIL`. In this case, the entire image must be in memory at the location specified by the `data` field. This field is used only by `ImageCodecBandDecompress` (page 868).

`port`

Discussion
Points to the color graphics port that receives the decompressed image.

`dstPixMap`

Discussion
Points to the pixel map where the decompressed image is to be displayed. The `GDevice` global variable is set to the destination graphics device. The contents of this pixel map differ from a standard pixel map in two ways. First, the `rowBytes` field is a full 16-bit value; the high-order bit is not necessarily set to 1. Second, the `baseAddr` field must contain a 32-bit clean address.

`maskBits`

Discussion
Contains an update mask. If your component can mask result data, use this mask to indicate which pixels in the destination pixel map to update. Your component indicates whether it can mask with the `codecCanMask` flag in the `flags` field of the `CodecCapabilities` structure referred to by the `capabilities` field. This field is updated in response to the `ImageCodecPreDecompress` (page 912) request. If the mask has not changed since the last `ImageCodecBandDecompress` request, the `codecConditionCodecChangedMask` flag in the `conditionFlags` field is set to 0. This field is used only by `ImageCodecBandDecompress` (page 868).

`mattePixMap`

Discussion
Points to a pixel map that contains a blend matte. The matte can be defined at any supported pixel depth; the matte depth need not correspond to the source or destination depths. The matte must be in the coordinate system of the source image. If the application does not want to apply a blend matte, this field is set to `NIL`. The contents of this pixel map differ from a standard pixel map in two ways. First, the `rowBytes` field is a full 16-bit value; the high-order bit is not necessarily set to 1. Second, the `baseAddr` field must contain a 32-bit clean address. This field is used only by `ImageCodecBandDecompress` (page 868).

`srcRect`

Discussion
Points to a rectangle defining the portion of the image to decompress. This rectangle must lie within the boundary rectangle of the compressed image, which is defined by the `width` and height fields of the image description structure referred to by the `imageDescription` field.

`matrix`

Discussion
Points to a matrix structure that specifies how to transform the image during decompression.

`accuracy`

Discussion
Constant (see below) that specifies the accuracy desired in the decompressed image. Values for this parameter are on the same scale as compression quality; see `CompressImage` (page 605). See these constants:

`codecMinQuality`

`codecLowQuality`

`codecNormalQuality`

`codecHighQuality`

`codecMaxQuality`

`codecLosslessQuality`

`transferMode`

Discussion
Specifies the QuickDraw transfer mode for the operation; see `Graphics Transfer Modes`.

`frameTime`

Discussion
Contains a pointer to an `ICMFrameTimeRecord` structure. This structure contains a frame's time information for scheduled asynchronous decompression operations.

`matrixFlags`

Discussion
Flag (see below) specifying the transformation matrix. Set to 0 for no transformation. See these constants:

`matrixFlagScale2x`

`matrixFlagScale1x`

`matrixFlagScaleHalf`

`matrixType`

Discussion
Contains the type of the transformation matrix, as returned by `GetMatrixType` (page 669).

`dstRect`

Discussion
The destination rectangle. It is the result of transforming the source rectangle (the `srcRect` parameter) by the transformation matrix (the `matrix` parameter).

`majorSourceChangeSeed`

Discussion
Contains an integer value that is incremented each time a data source is added or removed. This provides a fast way for a codec to know when it needs to redetermine which data source inputs are available.

`minorSourceChangeSeed`

Discussion
Contains an integer value that is incremented each time a data source is added or removed, or the data contained in any of the data sources changes. This provides a way for a codec to know if the data available to it has changed.

`sourceData`

Discussion
Contains a pointer to a `CDSequenceDataSource` structure. This structure contains a linked list of all data sources. Because each data source contains a link to the next data source, a codec can access all data sources from this field.

`maskRegion`

Discussion
If the `maskRegion` field is not `NIL`, it contains a QuickDraw region that is equivalent to the bit map contained in the `maskBits` field. For some codecs, using the QuickDraw region may be more convenient than the mask bit map.

`wantedDestinationPixelTypes`

Discussion
Filled in by the codec during the execution of `ImageCodecPreDecompress` (page 912). Contains a handle to a zero-terminated list of non-RGB pixels that the codec can decompress to. Leave set to `NIL` if the codec does not support non-RGB pixel spaces. The ICM copies this data structure, so it is up to the codec to dispose of it later. Since the predecompress call can be called often, it is suggested that codecs allocate this handle during the Open function and dispose of it during the Close function.

`screenFloodMethod`

Discussion
A constant (see below) for codecs that require key-color flooding. See these constants:

`kScreenFloodMethodNone`
`kScreenFloodMethodKeyColor`
`kScreenFloodMethodAlpha`

`screenFloodValue`

Discussion
If `screenFloodMethod` is `kScreenFloodMethodKeyColor`, contains the index of the color that should be used to flood the image area on screen when a refresh occurs. This is valid for both indexed and direct screen devices (e.g., for devices with 16 bit depth, it should contain the 5-5-5 RGB value). If `screenFloodMethod` is `kScreenFloodMethodAlpha`, contains the value that the alpha channel should be flooded with.

`preferredOffscreenPixelSize`

Discussion
Should be filled in `ImageCodecPreDecompress` (page 912) with the preferred depth of an offscreen buffer should the ICM have to create one. It is not guaranteed that an offscreen buffer will actually be of this depth. A codec should still be sure to specify what depths it can decompress to by using the `capabilities` field. A codec might use this field if if was capable of decompressing to several depths, but was faster decompressing to a particular depth.

`syncFrameTime`

Discussion
A pointer to an `ICMFrameTimeInfo` structure. This structure contains timing information about the display of the frame.

`needUpdateOnTimeChange`

Discussion
Undocumented

`enableBlackLining`

Discussion
If TRUE, indicates that the client has requested blacklining (displaying every other line of the image). Blacklining increases the speed of movie playback while decreasing the image quality.

`needUpdateOnSourceChange`

Discussion
Undocumented

`pad`

Discussion
Unused.

`unused`

Discussion
Unused.

`finalDestinationPort`

Discussion
Undocumented

`requestedBufferWidth`

Discussion
Specifies the width of the image buffer to use, in pixels. For this value to be used, the `codecWantsSpecialScaling` flag in `CodecCapabilities` must be set.

`requestedBufferHeight`

Discussion
Specifies the height of the image buffer to use, in pixels. For this value to be used, the `codecWantsSpecialScaling` flag in `CodecCapabilities` must be set.

`displayableAreaOfRequestedBuffer`

Discussion
This field can be used to prevent parts of the requested buffer from being displayed. When the `codecWantsSpecialScaling` flag is set, this rectangle can be filled in to indicate what portion of the requested buffer's width and height should be used. The buffer rectangle created by the requested buffer is always based at (0,0), so this coordinate system is also used by `displayableAreaOfRequestedBuffer`. If this field is not filled in, a default value of (0,0,0,0) is used, and the entire buffer is displayed. Use this field if you are experiencing edge problems with FlashPix images.

`requestedSingleField`

Discussion
Undocumented

`needUpdateOnNextIdle`

Discussion
Undocumented

`pad2`

Discussion
Unused.

`bufferGammaLevel`

Discussion
The gamma level of the data buffer.

`taskWeight`

Discussion
The preferred weight for multiprocessing tasks implementing this operation. You should assign a value by means of the Mac OS function `MPSetTaskWeight`.

`taskName`

Discussion
The preferred type for multiprocessing tasks implementing this operation. You should assign a value by means of the Mac OS function `MPSetTaskType`.

Discussion
The Image Compression Manager creates the decompression parameters structure, and your image decompressor component is required only to provide values for the `wantedDestinationPixelSize` and `wantedDestinationPixelTypes` fields of the structure. Your image decompressor component can also modify other fields if necessary. For example, if it can scale images, it must set the `codecCapabilityCanScale` flag in the `capabilities` field of the structure.

Version Notes
Some of the fields in `CodecDecompressParams` were added for various versions of QuickTime starting with version 2.1. See comments in the C interface file for details.

Related Functions
`ImageCodecBandDecompress` (page 868)
`ImageCodecBeginBand` (page 869)
`ImageCodecEffectBegin` (page 882)
`ImageCodecEffectSetup` (page 887)
`ImageCodecNewImageBufferMemory` (page 909)
`ImageCodecNewImageGWorld` (page 910)
`ImageCodecPreDecompress` (page 912)
`ImageCodecPreflight` (page 913)

Declared In
`ImageCodec.h`

ComponentMPWorkFunctionUPP

Represents a type used by the Image Codec API.

```
typedef STACK_UPP_TYPE(ComponentMPWorkFunctionProcPtr) ComponentMPWorkFunctionUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Components.h`

EffectsFrameParams

Contains information about the current frame of a video effect.

```
struct EffectsFrameParams {
     ICMFrameTimeRecord    frameTime;
     long                  effectDuration;
     Boolean               doAsync;
     unsigned char         pad[3];
     EffectSourcePtr       source;
     void *                refCon;
 };
```

Fields

`frameTime`

Discussion
Timing data for the current frame. This structure includes information such as the total number of frames being rendered in this sequence, and the current frame number.

`effectDuration`

Discussion
The duration of a single effect frame.

`doAsync`

Discussion
This field contains TRUE if the effect can process asynchronously.

`pad`

Discussion
Unused.

`source`

Discussion
A pointer to the input sources; see the `EffectSource` structure.

`refCon`

Discussion
A pointer to storage for this instantiation of the effect.

Related Functions
`ImageCodecEffectBegin` (page 882)
`ImageCodecEffectCancel` (page 882)
`ImageCodecEffectRenderFrame` (page 885)

Declared In
`ImageCodec.h`

EffectsFrameParamsPtr

Represents a type used by the Image Codec API.

```
typedef EffectsFrameParams * EffectsFrameParamsPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCodec.h`

EffectSource

Provides data for the EffectsFrameParams structure.

```
struct EffectSource {
     long              effectType;
     Ptr               data;
     SourceData        source;
     EffectSourcePtr   next;
```

Fields

`effectType`

Discussion

The type of the effect or a default effect type constant (see below). Enter `kEffectRawSource` if the source is raw image compression manager data. See these constants:

`kEffectRawSource`

`kEffectGenericType`

`data`

Discussion

A pointer to the track data for the effect.

`source`

Discussion

The source itself.

`next`

Discussion

A pointer to the next source in the input chain.

`lastTranslatedFrameTime`

Discussion

The start frame time of last converted frame; this value may be -1.

`lastFrameDuration`

Discussion

The duration of the last converted frame; this value may be 0.

`lastFrameTimeScale`

Discussion

The time scale of this source frame; this field has meaning only if the `lastTranslatedFrameTime` and `lastFrameDuration` fields are valid.

Related Functions

`ImageCodecEffectConvertEffectSourceToFormat` (page 883)

Declared In

`ImageCodec.h`

EffectSourcePtr

Represents a type used by the Image Codec API.

```
typedef EffectSource * EffectSourcePtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCodec.h`

gxPaths

Encapsulates a multiple-path geometry.

```
struct gxPaths {
     long      contours;
     gxPath    contour[1];
 };
```

Fields
`contours`

Discussion
The number of path contours.

`contour`

Discussion
The path contours; see `gxPath`.

Discussion
The contours field indicates the total number of contours (in other words, the total number of separate paths), and the `contour` field is an array that contains the path geometries. Since a `gxPaths` structure is of variable length and every element in it is of type `long`, you can define a path geometry as an array of long integer values.

Related Functions
`CurveCountPointsInPath` (page 856)
`CurveGetLength` (page 859)
`CurveGetNearestPathPoint` (page 859)
`CurveGetPathPoint` (page 860)
`CurveLengthToPoint` (page 862)
`CurvePathPointToLength` (page 864)
`CurveSetPathPoint` (page 865)

Declared In
`ImageCodec.h`

gxPoint

Defines a point in vector graphics.

```
struct gxPoint {
     Fixed    x;
     Fixed    y;
 };
```

Fields

`x`

Discussion

A horizontal distance. Greater values of the `x` field indicate distances further to the right.

`y`

Discussion

A vertical distance. Greater values of the `y` field indicate distances further down.

Discussion

The location of the origin depends on the context where you use the point; for example, it might be the upper-left corner of a view port. Notice that the `x` and y fields are of type `Fixed`. QuickDraw GX allows you to specify fractional coordinate positions.

Related Functions

`CurveGetPathPoint` (page 860)
`CurveInsertPointIntoPath` (page 861)
`CurveSetPathPoint` (page 865)

Declared In

`ImageCodec.h`

ImageCodecMPDrawBandUPP

Represents a type used by the Image Codec API.

```
typedef STACK_UPP_TYPE(ImageCodecMPDrawBandProcPtr) ImageCodecMPDrawBandUPP;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCodec.h`

ImageCodecTimeTriggerUPP

Represents a type used by the Image Codec API.

```
typedef STACK_UPP_TYPE(ImageCodecTimeTriggerProcPtr) ImageCodecTimeTriggerUPP;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCodec.h`

ImageSubCodecDecompressCapabilities

Returned by an image decompressor component in response to ImageCodecInitialize.

```
struct ImageSubCodecDecompressCapabilities {
     long       recordSize;
     long       decompressRecordSize;
     Boolean    canAsync;
     UInt8      pad0;
```

Fields

`recordSize`

Discussion

The size of this structure in bytes.

`decompressRecordSize`

Discussion

The size of the `ImageSubCodecDecompressRecord` structure that your image decompressor component requires. This structure is used to pass information from `ImageCodecBeginBand` (page 869) to `ImageCodecDrawBand` (page 880) and `ImageCodecEndBand` (page 888).

`canAsync`

Discussion

Specifies whether your image decompressor component can perform asynchronous scheduled decompression. This should be TRUE unless your image decompressor component calls functions that cannot be called during interrupt time.

`pad0`

Discussion

Unused.

Discussion

The first function call that your image decompressor component receives from the base image decompressor is always a call to `ImageCodecInitialize` (page 906). In response to this call, your image decompressor component returns an `ImageSubCodecDecompressCapabilities` structure that specifies its capabilities.

Related Functions

`ImageCodecInitialize` (page 906)

Declared In

`ImageCodec.h`

ImageSubCodecDecompressRecord

Contains information needed for decompressing a frame.

```
struct ImageSubCodecDecompressRecord {
     Ptr                        baseAddr;
     long                       rowBytes;
     Ptr                        codecData;
     ICMProgressProcRecord      progressProcRecord;
     ICMDataProcRecord          dataProcRecord;
     void *                     userDecompressRecord;
     UInt8                      frameType;
     UInt8                      pad[3];
     long                       priv[2];
 };
```

Fields

`baseAddr`

Discussion

The address of the destination pixel map, which includes adjustment for the offset. Note that if the bit depth of the pixel map is less than 8, your image decompressor component must adjust for the bit offset.

`rowBytes`

Discussion

The offset in bytes from one row of the destination pixel map to the next. The value of the `rowBytes` field must be less than 0x4000.

`codecData`

Discussion

A pointer to the data to be decompressed.

`progressProcRecord`

Discussion

An `ICMProgressProcRecord` structure that specifies a progress function. This function reports on the progress of a decompression operation. If there is no progress function, the Image Compression Manager sets the `progressProc` field in the `ICMProgressProcRecord` structure to `NIL`.

`dataProcRecord`

Discussion

An `ICMDataProcRecord` structure that specifies a data-loading function. If the data to be decompressed is not all in memory, your component can call this function to load more data. If there is no data-loading function, the Image Compression Manager sets the `dataProc` field in the `ICMDataProcRecord` structure to `NIL`, and the entire image must be in memory at the location specified by the `codecData` field of the `ImageSubCodecDecompressRecord` structure.

`userDecompressRecord`

Discussion

A pointer to storage for the decompression operation. The storage is allocated by the base image decompressor after it calls `ImageCodecInitialize` (page 906). The size of the storage is determined by the `decompressRecordSize` field of the `ImageSubCodecDecompressCapabilities` structure that is returned by `ImageCodecInitialize`. Your image decompressor component should use this storage to store any additional information needed about the frame in order to decompress it.

`frameType`

Discussion
A constant (see below) that indicates the `frame` type. See these constants:

`kCodecFrameTypeUnknown`

`kCodecFrameTypeKey`

`kCodecFrameTypeDifference`

`kCodecFrameTypeDroppableDifference`

`pad`

Discussion
Unused.

`priv`

Discussion
Private to QuickTime; do not use.

Related Functions
`ImageCodecBeginBand` (page 869)
`ImageCodecDrawBand` (page 880)
`ImageCodecEndBand` (page 888)
`ImageCodecMPDrawBandProc`

Declared In
`ImageCodec.h`

QTParameterValidationOptions

Represents a type used by the Image Codec API.

```
typedef long QTParameterValidationOptions;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCodec.h`

SMPTEFlags

Represents a type used by the Image Codec API.

```
typedef long SMPTEFlags;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCodec.h`

SMPTEFrameReference

Represents a type used by the Image Codec API.

```
typedef long SMPTEFrameReference;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCodec.h`

SMPTEWipeType

Represents a type used by the Image Codec API.

```
typedef unsigned long SMPTEWipeType;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCodec.h`

Constants

Codec Properties

Constants that represent the properties of codecs.

```
enum {
                                        /* The minimum data size for spooling in
or out data */
  codecMinimumDataSize          = 32768L
};
enum {
  codecConditionFirstBand       = 1L << 0,
  codecConditionLastBand        = 1L << 1,
  codecConditionFirstFrame      = 1L << 2,
  codecConditionNewDepth        = 1L << 3,
  codecConditionNewTransform    = 1L << 4,
  codecConditionNewSrcRect      = 1L << 5,
  codecConditionNewMask         = 1L << 6,
  codecConditionNewMatte        = 1L << 7,
  codecConditionNewTransferMode = 1L << 8,
  codecConditionNewClut         = 1L << 9,
  codecConditionNewAccuracy     = 1L << 10,
  codecConditionNewDestination  = 1L << 11,
  codecConditionFirstScreen     = 1L << 12,
  codecConditionDoCursor        = 1L << 13,
  codecConditionCatchUpDiff     = 1L << 14,
  codecConditionMaskMayBeChanged = 1L << 15,
  codecConditionToBuffer        = 1L << 16,
  codecConditionCodecChangedMask = 1L << 31
};
enum {
  codecInfoResourceType         = 'cdci', /* codec info resource type */
  codecInterfaceVersion        = 2    /* high word returned in component GetVersion
 */
};
enum {
  codecSuggestedBufferSentinel  = 'sent' /* codec public resource containing
suggested data pattern to put past end of data buffer */
};
enum {
  codecUsesOverlaySurface       = 1L << 0, /* codec uses overlay surface */
  codecImageBufferIsOverlaySurface = 1L << 1, /* codec image buffer is overlay
surface, the bits in the buffer are on the screen */
  codecSrcMustBeImageBuffer     = 1L << 2, /* codec can only source data from an
image buffer */
  codecImageBufferIsInAGPMemory = 1L << 4, /* codec image buffer is in AGP space,
 byte writes are OK */
  codecImageBufferIsInPCIMemory = 1L << 5, /* codec image buffer is across a PCI
bus; byte writes are bad */
  codecImageBufferMemoryFlagsValid = 1L << 6, /* set by
ImageCodecNewImageBufferMemory/NewImageGWorld to indicate that it set the AGP/PCI
 flags (supported in QuickTime 6.0 and later) */
  codecDrawsHigherQualityScaled = 1L << 7, /* codec will draw higher-quality image
 if it performs scaling (eg, wipe effect with border) */
  codecSupportsOutOfOrderDisplayTimes = 1L << 8, /* codec supports frames queued
in one order for display in a different order, eg, IPB content */
  codecSupportsScheduledBackwardsPlaybackWithDifferenceFrames = 1L << 9 /* codec
can use additional buffers to minimise redecoding during backwards playback */
};
```

Constants

`codecConditionFirstBand`
: An input flag that indicates if this is the first band in the frame. If this flag is set to 1, then your component is being called for the first time for the current frame.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCodec.h`.

`codecConditionLastBand`
: An input flag that indicates if this is the last band in the frame. If this flag is set to 1, then your component is being called for the last time for the current frame. If the `codecConditionFirstBand` flag is also set to 1, this is the only time the Image Compression Manager is calling your component for the current frame.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCodec.h`.

`codecConditionFirstFrame`
: An input flag that indicates that this is the first frame to be decompressed for this image sequence.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCodec.h`.

`codecConditionNewDepth`
: An input flag that indicates that the depth of the destination has changed for this image sequence.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCodec.h`.

`codecConditionNewTransform`
: An input flag that indicates that the transformation matrix has changed for this sequence.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCodec.h`.

`codecConditionNewSrcRect`
: An input flag that indicates that the source rectangle has changed for this sequence.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCodec.h`.

`codecConditionNewMatte`
: An input flag that indicates that the matte pixel map has changed for this sequence.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCodec.h`.

`codecConditionNewTransferMode`
: An input flag that indicates that the transfer mode has changed for this sequence.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCodec.h`.

`codecConditionNewClut`
: An input flag that indicates that the color lookup table has changed for this sequence.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCodec.h`.

`codecConditionNewAccuracy`
An input flag that indicates to the component that the `accuracy` parameter has changed for this sequence.

Available in Mac OS X v10.0 and later.

Declared in `ImageCodec.h`.

`codecConditionNewDestination`
An input flag that indicates to the component that the destination pixel map has changed for this sequence.

Available in Mac OS X v10.0 and later.

Declared in `ImageCodec.h`.

`codecConditionFirstScreen`
Indicates when the codec is decompressing an image to the first of multiple screens. That is, if the decompressed image crosses multiple screens, then the codec can look at this flag to determine if this is the first time an image is being decompressed for each of the screens to which it is being decompressed. A codec that depends on the `maskBits` field of this structure being a valid `RgnHandle` on `ImageCodecPreDecompress` (page 912) needs to know that in this case it is not able to clip images since the region handle is only passed for the first of the screens; clipping would be incorrect for the subsequent screen for that image.

Available in Mac OS X v10.0 and later.

Declared in `ImageCodec.h`.

`codecConditionDoCursor`
Set to 1 if the decompressor component should shield and unshield the cursor for the current decompression operation. This flag should be set only if the codec has indicated its ability to handle cursor shielding by setting the `codecCanShieldCursor` flag in the `capabilities` field during `ImageCodecPreDecompress` (page 912).

Available in Mac OS X v10.0 and later.

Declared in `ImageCodec.h`.

`codecConditionCatchUpDiff`
Indicates if the current frame is a "catch-up" frame. Set this flag to 1 if the current frame is a catch-up frame. Note that you must also set the `codecFlagCatchUpDiff` flag to 1. This may be useful to decompressors that can drop frames when playback is falling behind.

Available in Mac OS X v10.0 and later.

Declared in `ImageCodec.h`.

`codecConditionMaskMayBeChanged`
The Image Compression Manager has always included support for decompressors that could provide a bit mask of pixels that were actually drawn when a particular frame was decompressed. If a decompressor can provide a bit mask of pixels that changed, the Image Compression Manager transfers to the screen only the pixels that actually changed. QuickTime 2.1 extended this capability by adding this new condition flag. The decompressor should write back the mask only if this flag is set. This flag is used only by `ImageCodecFlush` (page 891).

Available in Mac OS X v10.0 and later.

Declared in `ImageCodec.h`.

`codecConditionToBuffer`
Set to 1 if the current decompression operation is decompressing into an offscreen buffer.

Available in Mac OS X v10.0 and later.

Declared in `ImageCodec.h`.

`codecConditionCodecChangedMask`
: An output flag that indicates that the component has changed the mask bits. If your image decompressor component can mask decompressed images and if some of the image pixels should not be written to the screen, set to 0 the corresponding bits in the mask defined by the `maskBits` field in the decompression parameter structure. In addition, set this flag to 1. Otherwise, set this flag to 0.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCodec.h`.

`codecInfoResourceType`
: Codec info resource type.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCodec.h`.

`codecInterfaceVersion`
: High word returned in component `GetVersion`.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCodec.h`.

`codecSuggestedBufferSentinel`
: Codec public resource containing suggested data pattern to put past end of data buffer.

  Available in Mac OS X v10.2 and later.

  Declared in `ImageCodec.h`.

`codecUsesOverlaySurface`
: *Undocumented*

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCodec.h`.

`codecImageBufferIsOverlaySurface`
: Indicates that the codec's image buffer is an overlay surface; the bits in the buffer appear on the screen.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCodec.h`.

`codecSrcMustBeImageBuffer`
: Indicates that the codec can accept source data only from an image buffer.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCodec.h`.

`codecImageBufferIsInAGPMemory`
: Indicates that the codec's image buffer resides in AGP address space and accepts byte writes.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCodec.h`.

`codecImageBufferIsInPCIMemory`
: Codec image buffer is across a PCI bus; byte writes are bad.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCodec.h`.

`codecImageBufferMemoryFlagsValid`
: Set by `ImageCodecNewImageBufferMemory` (page 909) or `NewImageGWorld` (page 694) to indicate that the `codecImageBufferIsInAGPMemory` and `codecImageBufferIsInPCIPMemory` flags have been set correctly.

  Available in Mac OS X v10.2 and later.

  Declared in `ImageCodec.h`.

`codecDrawsHigherQualityScaled`
: Indicates that the codec will draw a higher quality image if it performs scaling; for example, while drawing a wipe effect with a border.

  Available in Mac OS X v10.2 and later.

  Declared in `ImageCodec.h`.

`codecSupportsOutOfOrderDisplayTimes`
: Codec supports frames queued in one order for display in a different order, for example IPB content.

  Available in Mac OS X v10.3 and later.

  Declared in `ImageCodec.h`.

Declared In
`ImageCodec.h`

ImageSubCodecDecompressRecord Values

Constants passed to ImageSubCodecDecompressRecord.

```
enum {
  kCodecFrameTypeUnknown          = 0,
  kCodecFrameTypeKey              = 1,
  kCodecFrameTypeDifference       = 2,
  kCodecFrameTypeDroppableDifference = 3
};
```

Constants

`kCodecFrameTypeUnknown`
: The frame type is unknown.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCodec.h`.

`kCodecFrameTypeKey`
: This is a key frame.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCodec.h`.

`kCodecFrameTypeDifference`
: This is a difference frame.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCodec.h`.

Declared In
`ImageCodec.h`

EffectSource Values

Constants passed to EffectSource.

```
enum {
  kEffectRawSource              = 0,    /* the source is raw image data*/
  kEffectGenericType            = 'geff' /* generic effect for combining others*/
};
```

Constants

`kEffectRawSource`

The source is raw Image Compression Manager data.

Available in Mac OS X v10.0 and later.

Declared in `ImageCodec.h`.

Declared In

`ImageCodec.h`

ImageCodecValidateParameters Values

Constants passed to ImageCodecValidateParameters.

```
enum {
  kParameterValidationNoFlags    = 0x00000000,
  kParameterValidationFinalValidation = 0x00000001
};
```

Declared In

`ImageCodec.h`

CodecDecompressParams Values

Constants passed to CodecDecompressParams.

```
enum {
  kScreenFloodMethodNone        = 0,
  kScreenFloodMethodKeyColor    = 1,
  kScreenFloodMethodAlpha       = 2
};
enum {
  matrixFlagScale2x             = 1L << 7,
  matrixFlagScale1x             = 1L << 6,
  matrixFlagScaleHalf           = 1L << 5
};
```

Constants

`kScreenFloodMethodNone`

No method; value is 0.

Available in Mac OS X v10.0 and later.

Declared in `ImageCodec.h`.

`kScreenFloodMethodKeyColor`
: Key color method; value is 1.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCodec.h`.

`kScreenFloodMethodAlpha`
: Alpha channel method; value is 2.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCodec.h`.

`matrixFlagScale2x`
: Double-scale; value is `1L<<7`.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCodec.h`.

`matrixFlagScale1x`
: Single-scale; value is `1L<<6`.

  Available in Mac OS X v10.0 and later.

  Declared in `ImageCodec.h`.

Declared In

`ImageCodec.h`

Import and Export Reference for QuickTime

Framework:	Frameworks/QuickTime.framework
Declared in	ImageCompression.h

Overview

Image importers and exporters manage the import and export of graphic images, such as JPEG, TIFF, Photoshop, and PNG. Movie data exchange components support the import and export of other multimedia formats, such as AIFF, WAVE, AVI, MPEG-1, MIDI, MPEG-4, 3GPP, MP3, MPEG-2, H.263, and OpenDML. QuickTime can open any format file for which it has an importer and create any for which it has an exporter.

Functions by Task

Accessing a Graphics Exporter's Input Image

`GraphicsExportDrawInputImage` (page 978)
Draws a rectangular portion of the input image in a graphics export operation.

`GraphicsExportGetInputImageDepth` (page 989)
Returns the depth of the input image for a graphics export operation.

`GraphicsExportGetInputImageDescription` (page 989)
Returns an image description describing the input image in a graphics export operation.

`GraphicsExportGetInputImageDimensions` (page 990)
Returns the dimensions of the input image in a graphics export operation.

Accessing Graphics Exporter Settings

`GraphicsExportGetColorSyncProfile` (page 979)
Gets the current value of the ColorSync profile for a graphics export operation.

`GraphicsExportGetCompressionMethod` (page 980)
Returns the compression method for a graphics export operation.

`GraphicsExportGetCompressionQuality` (page 980)
Returns the compression quality value for a graphics export operation.

`GraphicsExportGetDepth` (page 982)
Returns the current depth setting for a graphics export operation.

`GraphicsExportGetDontRecompress` (page 983)
: Determines whether the original compressed data for a graphics export operation will not be decompressed and recompressed, but be copied through to the output file.

`GraphicsExportGetInterlaceStyle` (page 994)
: Returns the interlace style in a graphics export operation.

`GraphicsExportGetMetaData` (page 994)
: Returns the current user data setting in a graphics export operation.

`GraphicsExportGetResolution` (page 1000)
: Determines the resolution of a graphics exporter component.

`GraphicsExportGetTargetDataSize` (page 1001)
: Returns the current desired maximum data size for a graphics export operation.

`GraphicsExportSetColorSyncProfile` (page 1006)
: Sets the ColorSync profile to embed in the image file for a graphics export operation.

`GraphicsExportSetCompressionMethod` (page 1006)
: Defines the compression method to use in a graphics export operation.

`GraphicsExportSetCompressionQuality` (page 1007)
: Defines the compression quality for a graphics export operation.

`GraphicsExportSetDepth` (page 1008)
: Defines the depth to use in a graphics export operation.

`GraphicsExportSetDontRecompress` (page 1008)
: Requests that the original compressed data for a graphics export operation not be decompressed and recompressed, but be copied through to the output file.

`GraphicsExportSetInterlaceStyle` (page 1018)
: Defines the interlace style for a graphics export operation.

`GraphicsExportSetMetaData` (page 1018)
: Defines supplemental data for a graphics export operation, such as copyright text.

`GraphicsExportSetResolution` (page 1023)
: Defines the resolution to store in the image file for a graphics export operation.

`GraphicsExportSetTargetDataSize` (page 1025)
: Defines a desired maximum data size for a graphics export operation and asks for a quality that does not exceed that size.

Drawing Imported Images

`GraphicsImportDraw` (page 1030)
: Draws an imported image.

`GraphicsImportGetGWorld` (page 1050)
: Returns the current graphics port and device for drawing an imported image.

`GraphicsImportSetGWorld` (page 1072)
: Sets the graphics port and device for drawing an imported image.

Finding Out About Graphics Export Image Formats

`GraphicsExportGetDefaultFileNameExtension` (page 981)
Returns the suggested file name extension for a graphics export operation.

`GraphicsExportGetDefaultFileTypeAndCreator` (page 982)
Returns the suggested file type and creator for a graphics export operation.

`GraphicsExportGetMIMETypeList` (page 995)
Returns MIME types and other information about the graphics format in a graphics export operation.

Getting and Setting Progress Procs

`GraphicsExportGetProgressProc` (page 999)
Returns the current progress function for a graphics export operation.

`GraphicsExportSetProgressProc` (page 1023)
Installs a progress function in a graphics export operation.

Getting Image Characteristics

`GraphicsImportDoesDrawAllPixels` (page 1027)
Asks whether the graphics importer expects to draw every pixel.

`GraphicsImportGetImageDescription` (page 1052)
Returns image description information for an imported image.

`GraphicsImportGetMetaData` (page 1054)
Extracts user data from an imported image file.

`GraphicsImportGetNaturalBounds` (page 1055)
Returns the bounding rectangle of an imported image.

`GraphicsImportValidate` (page 1078)
Validates image data for a data reference to an imported image.

Getting MIME Types

`GraphicsImportGetMIMETypeList` (page 1055)
Returns a list of MIME types supported by the graphics importer component.

Internal Graphics Export Routines

`GraphicsExportCanTranscode` (page 974)
Asks whether the current graphics export operation should be performed by transcoding.

`GraphicsExportCanUseCompressor` (page 975)
Asks whether to use a compressor in a graphics export operation.

`GraphicsExportDoStandaloneExport` (page 976)
Performs a standalone graphics export operation.

`GraphicsExportDoTranscode` (page 977)
Performs a graphics export operation by transcoding.

`GraphicsExportDoUseCompressor` (page 977)
Performs a graphics export operation with compression.

Managing Graphics Importers

`GraphicsImportGetColorSyncProfile` (page 1037)
Returns a ColorSync profile for an imported image, if one is embedded in the image file.

`GraphicsImportGetDataOffsetAndSize` (page 1039)
Returns the offset and size of the compressed image data within an imported image file.

`GraphicsImportGetDataOffsetAndSize64` (page 1040)
Provides a 64-bit version of GraphicsImportGetDataOffsetAndSize.

`GraphicsImportGetDataReferenceOffsetAndLimit` (page 1041)
Returns the data reference starting offset and data size limit for an imported image.

`GraphicsImportGetDataReferenceOffsetAndLimit64` (page 1042)
Provides a 64-bit version of GraphicsImportGetDataReferenceOffsetAndLimit.

`GraphicsImportGetDefaultClip` (page 1043)
Returns the default clipping region for an imported image, if one is stored there.

`GraphicsImportGetDefaultGraphicsMode` (page 1044)
Returns the default graphics mode for an imported image, if one is stored there.

`GraphicsImportGetDefaultMatrix` (page 1045)
Returns the default matrix for an imported image, if one is stored there.

`GraphicsImportGetDefaultSourceRect` (page 1045)
Returns the default source rectangle for an imported image, if one is stored there.

`GraphicsImportGetDestRect` (page 1046)
Returns the destination rectangle for an imported image.

`GraphicsImportGetFlags` (page 1049)
Returns the current flags of a graphics importer component.

`GraphicsImportGetImageCount` (page 1051)
Returns the number of images in an imported image file.

`GraphicsImportGetImageIndex` (page 1053)
Returns the current image index for an imported image.

`GraphicsImportReadData` (page 1059)
Reads imported image data.

`GraphicsImportReadData64` (page 1060)
Provides a 64-bit version of GraphicsImportReadData.

`GraphicsImportSetDataReferenceOffsetAndLimit` (page 1068)
Specifies the data reference starting offset and data size limit for an imported image.

`GraphicsImportSetDataReferenceOffsetAndLimit64` (page 1069)
Provides a 64-bit version of GraphicsImportSetDataReferenceOffsetAndLimit.

`GraphicsImportSetDestRect` (page 1070)
Sets the destination rectangle for a graphics import operation.

`GraphicsImportSetFlags` (page 1071)
: Sets the flags for a graphics importer component.

`GraphicsImportSetImageIndex` (page 1073)
: Specifies the image index for an imported image.

`GraphicsImportSetImageIndexToThumbnail` (page 1074)
: Looks for a graphics subimage that contains a thumbnail.

Obtaining Graphics Exporter Settings

`GraphicsExportGetSettingsAsAtomContainer` (page 1000)
: Retrieves the current settings from a graphics exporter component.

`GraphicsExportGetSettingsAsText` (page 1001)
: Retrieves the current settings from the graphics export component in a user-readable format.

`GraphicsExportRequestSettings` (page 1005)
: Displays a dialog for the user to configure graphics exporter settings, if applicable.

`GraphicsExportSetSettingsFromAtomContainer` (page 1024)
: Sets the graphics exporter component's current configuration to match the settings in a passed atom container.

Reading Graphics Exporter Input Data

`GraphicsExportGetInputDataSize` (page 986)
: Returns the number of bytes of original image data that can be read in a graphics export operation.

`GraphicsExportMayExporterReadInputData` (page 1002)
: Asks whether the image source for a graphics export operation is in a form that the exporter can read.

`GraphicsExportReadInputData` (page 1003)
: Reads the original image data in a graphics export operation.

Restricting the Range of an Input Image's Source

`GraphicsExportGetInputOffsetAndLimit` (page 991)
: Retrieves the current input offset and limit in a graphics export operation.

`GraphicsExportSetInputOffsetAndLimit` (page 1015)
: Specifies the portion of an input data reference, file, handle or pointer that a graphics exporter is permitted to read.

Saving Image Files

`GraphicsImportDoExportImageFileDialog` (page 1028)
: Presents a dialog box letting the user save an imported image in a foreign file format.

`GraphicsImportExportImageFile` (page 1031)
: Saves an imported image in a foreign file format.

`GraphicsImportGetAsPicture` (page 1034)
    Creates a QuickDraw picture handle to an imported image.

`GraphicsImportGetExportImageTypeList` (page 1047)
    Returns information about available export formats for a graphics importer.

`GraphicsImportGetExportSettingsAsAtomContainer` (page 1048)
    Retrieves settings for image files exported by the graphics importer.

`GraphicsImportSaveAsPicture` (page 1060)
    Creates a QuickDraw picture file for an imported image.

`GraphicsImportSaveAsQuickTimeImageFile` (page 1062)
    Creates a QuickTime Image file of an imported image.

`GraphicsImportSetExportSettingsFromAtomContainer` (page 1070)
    Determines settings for the export of imported image files.

Setting Drawing Parameters

`GraphicsImportGetBoundsRect` (page 1035)
    Returns the bounding rectangle for drawing an imported image.

`GraphicsImportGetClip` (page 1036)
    Returns the current clipping region for an imported image.

`GraphicsImportGetGraphicsMode` (page 1050)
    Returns the graphics transfer mode for an imported image.

`GraphicsImportGetMatrix` (page 1053)
    Returns the transformation matrix to be used for drawing an imported image.

`GraphicsImportGetProgressProc` (page 1057)
    Returns the current progress function for a graphics import operation.

`GraphicsImportGetQuality` (page 1057)
    Returns the image quality value for an imported image.

`GraphicsImportGetSourceRect` (page 1058)
    Returns the current source rectangle for an imported image.

`GraphicsImportSetBoundsRect` (page 1063)
    Defines the rectangle in which to draw an imported image.

`GraphicsImportSetClip` (page 1064)
    Defines the clipping region for drawing an imported image.

`GraphicsImportSetGraphicsMode` (page 1072)
    Sets the graphics transfer mode for an imported image.

`GraphicsImportSetMatrix` (page 1075)
    Defines the transformation matrix to use for drawing an imported image.

`GraphicsImportSetProgressProc` (page 1076)
    Installs a progress procedure to call while drawing an imported image.

`GraphicsImportSetQuality` (page 1077)
    Sets the image quality value for an imported image.

`GraphicsImportSetSourceRect` (page 1078)
    Sets the source rectangle to use for an imported image.

Specifying a Graphics Import Data Source

`GraphicsImportGetDataFile` (page 1037)
: Returns the file containing the graphics data for an imported image.

`GraphicsImportGetDataHandle` (page 1038)
: Returns a handle to imported graphics data.

`GraphicsImportGetDataReference` (page 1040)
: Returns a data reference to imported graphics data.

`GraphicsImportSetDataFile` (page 1065)
: Specifies the file that contains imported graphics data.

`GraphicsImportSetDataHandle` (page 1066)
: Specifies the handle that references imported graphics data.

`GraphicsImportSetDataReference` (page 1067)
: Specifies the data reference for imported graphics data.

Specifying Destinations for Output Images

`GraphicsExportGetOutputDataReference` (page 995)
: Gets the output data reference handle in a graphics export operation.

`GraphicsExportGetOutputFile` (page 996)
: Returns the current output file for a graphics export operation.

`GraphicsExportGetOutputFileTypeAndCreator` (page 997)
: Gets the type and creator codes for the output file in a graphics export operation.

`GraphicsExportGetOutputHandle` (page 997)
: Returns the current output handle for a graphics export operation.

`GraphicsExportGetOutputOffsetAndMaxSize` (page 998)
: Returns the output starting offset and maximum size limit for a graphics export operation.

`GraphicsExportSetOutputDataReference` (page 1019)
: Returns the current output data reference for a graphics export operation.

`GraphicsExportSetOutputFile` (page 1020)
: Defines the output file for a graphics export operation.

`GraphicsExportSetOutputFileTypeAndCreator` (page 1020)
: Sets the file type and creator codes for the output file of a graphics export operation.

`GraphicsExportSetOutputHandle` (page 1021)
: Sets a handle to the output of a graphics export operation.

`GraphicsExportSetOutputOffsetAndMaxSize` (page 1022)
: Specifies the output starting offset and maximum size limit for a graphics export operation.

Specifying Sources for Graphics Exporter Input Images

`GraphicsExportGetInputDataReference` (page 985)
: Returns the current input data reference for a graphics export operation.

`GraphicsExportGetInputFile` (page 986)
: Returns the current input file for a graphics export operation.

`GraphicsExportGetInputGraphicsImporter` (page 987)
: Returns the current input graphics importer instance for a graphics export operation.

`GraphicsExportGetInputGWorld` (page 988)
: Returns the current input graphics world for a graphics export operation.

`GraphicsExportGetInputHandle` (page 988)
: Returns the current input handle for a graphics export operation.

`GraphicsExportGetInputPicture` (page 991)
: Returns the current input picture in a graphics export operation.

`GraphicsExportGetInputPixmap` (page 992)
: Returns the current input pixmap in a graphics export operation.

`GraphicsExportGetInputPtr` (page 993)
: Returns the current input pointer in a graphics export operation.

`GraphicsExportSetInputDataReference` (page 1011)
: Specifies that the source image for a graphics export operation is a compressed image stored in a data reference.

`GraphicsExportSetInputFile` (page 1011)
: Specifies that the source image for a graphics export operation is a compressed image stored in a file.

`GraphicsExportSetInputGraphicsImporter` (page 1012)
: Specifies that the source image for a graphics export operation is to be drawn by a graphics importer instance.

`GraphicsExportSetInputGWorld` (page 1013)
: Specifies that the source image for a graphics export operation is a graphics world.

`GraphicsExportSetInputHandle` (page 1014)
: Specifies that the source image for a graphics export operation is a compressed image referenced by a handle.

`GraphicsExportSetInputPicture` (page 1015)
: Specifies that the source image for a graphics export operation is a picture.

`GraphicsExportSetInputPixmap` (page 1016)
: Specifies that the source image for a graphics export operation is a pixmap.

`GraphicsExportSetInputPtr` (page 1017)
: Specifies that the source image for a graphics export operation is a compressed image stored at a fixed address in memory.

Working With Exif Files

`GraphicsExportGetExifEnabled` (page 983)
: Returns the graphics exporter's current Exif export setting.

`GraphicsExportGetThumbnailEnabled` (page 1002)
: Returns the current thumbnail creation setting for the graphics exporter when exporting Exif files.

`GraphicsExportSetExifEnabled` (page 1009)
: Determines whether or not the graphics exporter component should create Exif files.

`GraphicsExportSetThumbnailEnabled` (page 1025)
Determines whether or not the graphics exporter component should create an embedded thumbnail inside an exported Exif file.

Writing Graphics Exporter Output Data

`GraphicsExportGetOutputMark` (page 998)
Returns the current file position for a graphics export operation.

`GraphicsExportReadOutputData` (page 1004)
Reads output image data in a graphics export operation.

`GraphicsExportSetOutputMark` (page 1021)
Seeks to the specified file position in a graphics export operation.

`GraphicsExportWriteOutputData` (page 1026)
Writes output image data in a graphics export operation.

Supporting Functions

`GraphicsExportDoExport` (page 975)
Performs a graphics export operation.

`GraphicsExportGetInputCGBitmapContext` (page 984)
Retrieves the CGBitmapContext that the graphics exporter is using as its input image.

`GraphicsExportGetInputCGImage` (page 984)
Determines which Core Graphics CGImage is the source for a graphics export operation.

`GraphicsExportSetInputCGBitmapContext` (page 1010)
Sets the CGBitmapContext that the graphics exporter will use as its input image.

`GraphicsExportSetInputCGImage` (page 1010)
Specifies a Core Graphics CGImage as the source for a graphics export operation.

`GraphicsImportCreateCGImage` (page 1027)
Imports an image as a Core Graphics CGImage.

`GraphicsImportDoExportImageFileToDataRefDialog` (page 1029)
Presents a dialog box that lets the user save an imported image in a foreign file format.

`GraphicsImportExportImageFileToDataRef` (page 1033)
Saves an imported image in a foreign file format.

`GraphicsImportGetAliasedDataReference` (page 1033)
Deprecated.

`GraphicsImportGetBaseDataOffsetAndSize64` (page 1035)
Undocumented

`GraphicsImportGetDestinationColorSyncProfileRef` (page 1046)
Retrieves a ColorSync profile from a graphics importer component.

`GraphicsImportGetGenericColorSyncProfile` (page 1049)
Retrieves the generic colorsync profile for a graphics importer component.

`GraphicsImportGetOverrideSourceColorSyncProfileRef` (page 1056)
Retrieves the override ColorSync profile for a graphics importer component.

`GraphicsImportSaveAsPictureToDataRef` (page 1061)
: Creates a storage location that contains a QuickDraw picture for an imported image.

`GraphicsImportSaveAsQuickTimeImageFileToDataRef` (page 1063)
: Creates a storage location that contains a QuickTime image of an imported image.

`GraphicsImportSetDestinationColorSyncProfileRef` (page 1069)
: Sets the ColorSync profile for a graphics importer component.

`GraphicsImportSetOverrideSourceColorSyncProfileRef` (page 1075)
: Sets the override ColorSync profile for a graphics importer component.

`GraphicsImportWillUseColorMatching` (page 1079)
: Asks whether GraphicsImportDraw will use color matching if called with the current importer settings.

Functions

GraphicsExportCanTranscode

Asks whether the current graphics export operation should be performed by transcoding.

```
ComponentResult GraphicsExportCanTranscode (
   GraphicsExportComponent ci,
   Boolean *canTranscode
);
```

Parameters

ci
: The component instance that identifies your connection to the graphics exporter component.

canTranscode
: Points to a Boolean to receive the answer. TRUE means that the current export operation should be performed by transcoding, FALSE that it should not.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Graphics exporters may be able to transcode from some inputs and not from others. For instance, the JPEG graphics exporter is able to transcode compressed JPEG streams, but not other kinds of compressed data. The base graphics exporter makes this call to the format-specific graphics exporter to ask whether the current export operation should be done by transcoding. If the format-specific exporter replies that it should, the base exporter calls `GraphicsExportDoTranscode` (page 977) to do so. If the answer is no, then the format-specific exporter will not be able to transcode.

Special Considerations

This function is used for internal communication between the base and format-specific graphics exporter. Applications will not usually need to call it. Format-specific exporters may delegate this call, in which case the base graphics exporter's implementation gives a reply of FALSE.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GraphicsExportCanUseCompressor

Asks whether to use a compressor in a graphics export operation.

```
ComponentResult GraphicsExportCanUseCompressor (
   GraphicsExportComponent ci,
   Boolean *canUseCompressor,
   void *codecSettingsAtomContainerPtr
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

canUseCompressor

A Boolean variable to receive the answer.

codecSettingsAtomContainerPtr

A pointer to a `QTAtomContainer` variable. If the `canUseCompressor` parameter returns TRUE, the format-specific exporter should create a new QuickTime atom container with information about the compression operation and return it here.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The base graphics exporter makes this call of the format-specific graphics exporter to ask whether the current export operation should be done by using an image compressor. If the answer is TRUE, the format-specific exporter must also create and return an atom container. This atom container must contain a big-endian `'vide'` atom with at least a child atom of type `'sptl'` containing a `SCSpatialSettings` record specifying which compressor to use, the depth, and the spatial quality.

Special Considerations

This function is used for internal communication between the base and format-specific graphics exporter. Applications will not usually need to call it. Format-specific exporters may delegate this call, in which case the base graphics exporter's implementation gives a reply of FALSE.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GraphicsExportDoExport

Performs a graphics export operation.

```
ComponentResult GraphicsExportDoExport (
   GraphicsExportComponent ci,
   unsigned long *actualSizeWritten
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

actualSizeWritten

Points to a variable to receive the number of bytes written. If you are not interested in this information, pass `NIL`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Before calling this function , you must specify an input image, using one of the `GraphicsExportSetInput`... functions, and a destination for the output image file, using one of the `GraphicsExportSetOutput`... functions.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Graphic Import-Export

ImproveYourImage

qtgraphics.win

TextNameTool

ThreadsExporter

Declared In

`ImageCompression.h`

GraphicsExportDoStandaloneExport

Performs a standalone graphics export operation.

```
ComponentResult GraphicsExportDoStandaloneExport (
   GraphicsExportComponent ci
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

If both `GraphicsExportCanTranscode` (page 974) and `GraphicsExportCanUseCompressor` (page 975) reply FALSE, the base graphics exporter makes this call of the format-specific exporter to perform the export.

Special Considerations

This function is used for internal communication between the base and format-specific graphics exporter. Applications will not usually need to call it.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GraphicsExportDoTranscode

Performs a graphics export operation by transcoding.

```
ComponentResult GraphicsExportDoTranscode (
   GraphicsExportComponent ci
);
```

Parameters

`ci`

The component instance that identifies your connection to the graphics exporter component.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The base graphics exporter makes this call of the format-specific graphics exporter to perform a transcoding export. This function should call `GraphicsExportGetInputDataSize` (page 986) and `GraphicsExportReadInputData` (page 1003) to measure and read the input image data, and `GraphicsExportWriteOutputData` (page 1026) to write the output image file.

Special Considerations

This function is used for internal communication between the base and format-specific graphics exporter. Applications will not usually need to call it.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GraphicsExportDoUseCompressor

Performs a graphics export operation with compression.

```
ComponentResult GraphicsExportDoUseCompressor (
   GraphicsExportComponent ci,
   void *codecSettingsAtomContainer,
   ImageDescriptionHandle *outDesc
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

codecSettingsAtomContainer

An atom container returned by `GraphicsExportCanUseCompressor` (page 975).

outDesc

Points to an image description handle to receive an `ImageDescription` structure describing the compressed image.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The base graphics exporter makes this call to perform a compressing export.

Special Considerations

This function is used for internal communication between the base and format-specific graphics exporter. Applications will not usually need to call it. Format-specific exporters will normally delegate this call, unless they implement export to a container format like PICT or QuickTime Image. In that case, they will wrap the base exporter's implementation in one that forms the container about the compressed data.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GraphicsExportDrawInputImage

Draws a rectangular portion of the input image in a graphics export operation.

```
ComponentResult GraphicsExportDrawInputImage (
   GraphicsExportComponent ci,
   CGrafPtr gw,
   GDHandle gd,
   const Rect *srcRect,
   const Rect *dstRect
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

gw

A pointer to an offscreen graphics world, color graphics port, or basic graphics port.

gd

A handle to a `GDevice` record. If you pass a pointer to an offscreen graphics world in the `gw` parameter, set this parameter to `NIL` because `GraphicsExportDrawInputImage` ignores this parameter and sets the current device to the device attached to the offscreen graphics world.

srcRect

Specifies a portion of the input image.

dstRect

Specifies where in the drawing environment to draw that portion of the input image.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The `gw` and `gd` parameters specify a drawing environment such as you might pass to `GraphicsExportSetInputGWorld` (page 1013). The `srcRect` and `dstRect` boundaries need not be the same width and height; you can use this function to scale the `srcRect` image portion. This would be useful, for example, if you were writing a graphics exporter for a multiple-resolution format.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

ElectricImageComponent

ElectricImageComponent.win

Declared In

`ImageCompression.h`

GraphicsExportGetColorSyncProfile

Gets the current value of the ColorSync profile for a graphics export operation.

```
ComponentResult GraphicsExportGetColorSyncProfile (
   GraphicsExportComponent ci,
   Handle *colorSyncProfile
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

colorSyncProfile

Points to a variable to receive the ColorSync profile as a newly allocated handle.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Special Considerations

The caller is responsible for disposing of the returned handle.

Version Notes

Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GraphicsExportGetCompressionMethod

Returns the compression method for a graphics export operation.

```
ComponentResult GraphicsExportGetCompressionMethod (
   GraphicsExportComponent ci,
   long *compressionMethod
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

compressionMethod

Points to a value to receive the compression method.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GraphicsExportGetCompressionQuality

Returns the compression quality value for a graphics export operation.

```
ComponentResult GraphicsExportGetCompressionQuality (
   GraphicsExportComponent ci,
   CodecQ *spatialQuality
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

spatialQuality

Points to a variable to receive a quality constant (see below). See these constants:

`codecMinQuality`

`codecLowQuality`

`codecNormalQuality`

`codecHighQuality`

`codecMaxQuality`

`codecLosslessQuality`

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GraphicsExportGetDefaultFileNameExtension

Returns the suggested file name extension for a graphics export operation.

```
ComponentResult GraphicsExportGetDefaultFileNameExtension (
   GraphicsExportComponent ci,
   OSType *fileNameExtension
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

fileNameExtension

Points to a location to receive the file name extension.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
File name extensions are returned as upper-case big-endian four-character codes. For example, the extension `.png` would be returned as `'PNG '` (0x504E4720).

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
ThreadsExporter

Declared In
`ImageCompression.h`

GraphicsExportGetDefaultFileTypeAndCreator

Returns the suggested file type and creator for a graphics export operation.

```
ComponentResult GraphicsExportGetDefaultFileTypeAndCreator (
   GraphicsExportComponent ci,
   OSType *fileType,
   OSType *fileCreator
);
```

Parameters

ci
: The component instance that identifies your connection to the graphics exporter component.

fileType
: Points to a location to receive the suggested file type for the image file format. If you don't need this information, pass `NIL`.

fileCreator
: Points to a location to receive the suggested file creator for the new image file format. If you don't need this information, pass `NIL`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function, along with `GraphicsExportGetDefaultFileNameExtension` (page 981) and `GraphicsExportGetMIMETypeList` (page 995), returns information about the image format supported by a graphics exporter. Format-specific exporters must implement all three of these calls.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GraphicsExportGetDepth

Returns the current depth setting for a graphics export operation.

```
ComponentResult GraphicsExportGetDepth (
   GraphicsExportComponent ci,
   long *depth
);
```

Parameters

ci
: The component instance that identifies your connection to the graphics exporter component.

depth
: Points to a variable to receive the depth.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GraphicsExportGetDontRecompress

Determines whether the original compressed data for a graphics export operation will not be decompressed and recompressed, but be copied through to the output file.

```
ComponentResult GraphicsExportGetDontRecompress (
   GraphicsExportComponent ci,
   Boolean *dontRecompress
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

dontRecompress

Points to a Boolean to receive the recompression setting.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Even though it is not decompressed and recompressed, graphics data may be modified when it is copied through.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GraphicsExportGetExifEnabled

Returns the graphics exporter's current Exif export setting.

```
ComponentResult GraphicsExportGetExifEnabled (
   GraphicsExportComponent ci,
   Boolean *exifEnabled
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component. This function is supported only by the TIFF and JPEG graphics exporters.

exifEnabled

Pass a pointer to a variable that will be set to TRUE if `Exif` export is enabled.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 6.

Availability

Available in Mac OS X v10.1 and later.

Declared In

`ImageCompression.h`

GraphicsExportGetInputCGBitmapContext

Retrieves the CGBitmapContext that the graphics exporter is using as its input image.

```
ComponentResult GraphicsExportGetInputCGBitmapContext (
   GraphicsExportComponent ci,
   CGContextRef *bitmapContextRefOut
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

bitmapContextRef

A reference to the Core Graphics context.

Return Value

See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 6.4.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`ImageCompression.h`

GraphicsExportGetInputCGImage

Determines which Core Graphics CGImage is the source for a graphics export operation.

```
ComponentResult GraphicsExportGetInputCGImage (
   GraphicsExportComponent ci,
   CGImageRef *imageRefOut
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

imageRef

A reference to a Core Graphics image.

Return Value

See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 6.4.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`ImageCompression.h`

GraphicsExportGetInputDataReference

Returns the current input data reference for a graphics export operation.

```
ComponentResult GraphicsExportGetInputDataReference (
   GraphicsExportComponent ci,
   Handle *dataRef,
   OSType *dataRefType
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

dataRef

Points to a variable to receive the data reference handle.

dataRefType

Points to a variable to receive the `data` reference type.

Return Value

See `Error Codes`. If the current source is not a data reference, the function returns `paramErr`. The function returns `noErr` if there is no error.

Discussion

You can use this function to get the source of a graphics export operation. The source can be a QuickTime graphics importer component instance, a QuickDraw `Picture`, a graphics world, a `PixMap` structure, or a piece of compressed data described by an `ImageDescription` structure. Compressed data can be in a file, handle, pointer, or other data reference. The application must make sure that the source is not disposed of before the graphics exporter instance is closed or given a new source. All of the get and set functions for these sources are implemented by the base graphics exporter; format-specific importers should delegate all of them.

Special Considerations

The caller is responsible for disposing of the returned data reference handle.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GraphicsExportGetInputDataSize

Returns the number of bytes of original image data that can be read in a graphics export operation.

```
ComponentResult GraphicsExportGetInputDataSize (
   GraphicsExportComponent ci,
   unsigned long *size
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

size

Points to a variable to receive the size in bytes.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function is used by format-specific graphics exporters when transcoding. Applications will not normally need to call this function.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GraphicsExportGetInputFile

Returns the current input file for a graphics export operation.

```
ComponentResult GraphicsExportGetInputFile (
   GraphicsExportComponent ci,
   FSSpec *theFile
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

theFile

A pointer to the file specification of the file containing the graphics data.

Return Value
See `Error Codes`. Returns `noErr` if there is no error. If the current source is not a file, the function returns `paramErr`.

Discussion
You can use this function to get the source of a graphics export operation. The source can be a QuickTime graphics importer component instance, a QuickDraw `Picture`, a graphics world, a `PixMap` structure, or a piece of compressed data described by an `ImageDescription` structure. Compressed data can be in a file, handle, pointer, or other data reference. The application must make sure that the source is not disposed of before the graphics exporter instance is closed or given a new source. All of the get and set functions for these sources are implemented by the base graphics exporter; format-specific importers should delegate all of them.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GraphicsExportGetInputGraphicsImporter

Returns the current input graphics importer instance for a graphics export operation.

```
ComponentResult GraphicsExportGetInputGraphicsImporter (
   GraphicsExportComponent ci,
   GraphicsImportComponent *grip
);
```

Parameters

`ci`
    The component instance that identifies your connection to the graphics exporter component.

`grip`
    Points to a variable to receive the source graphics importer.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
You must get the source of a graphics export operation. The source can be a QuickTime graphics importer component instance, a QuickDraw `Picture`, a graphics world, a `PixMap` structure, or a piece of compressed data described by an `ImageDescription` structure. Compressed data can be in a file, handle, pointer, or other data reference. The application must make sure that the source is not disposed of before the graphics exporter instance is closed or given a new source. All of the get and set functions for these sources are implemented by the base graphics exporter; format-specific importers should delegate all of them.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GraphicsExportGetInputGWorld

Returns the current input graphics world for a graphics export operation.

```
ComponentResult GraphicsExportGetInputGWorld (
   GraphicsExportComponent ci,
   GWorldPtr *gworld
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

gworld

Points to a variable to receive the source graphics world.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
You can use this function to get the source of a graphics export operation. The source can be a QuickTime graphics importer component instance, a QuickDraw `Picture`, a graphics world, a `PixMap` structure, or a piece of compressed data described by an `ImageDescription` structure. Compressed data can be in a file, handle, pointer, or other data reference. The application must make sure that the source is not disposed of before the graphics exporter instance is closed or given a new source. All of the get and set functions for these sources are implemented by the base graphics exporter; format-specific importers should delegate all of them.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GraphicsExportGetInputHandle

Returns the current input handle for a graphics export operation.

```
ComponentResult GraphicsExportGetInputHandle (
   GraphicsExportComponent ci,
   Handle *h
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

h

A pointer to receive the handle.

Return Value
See `Error Codes`. Returns `noErr` if there is no error. If the current source is not a handle, the function returns `paramErr`.

Discussion
You can use this function to get the source of a graphics export operation. The source can be a QuickTime graphics importer component instance, a QuickDraw `Picture`, a graphics world, a `PixMap` structure, or a piece of compressed data described by an `ImageDescription` structure. Compressed data can be in a file, handle, pointer, or other data reference. The application must make sure that the source is not disposed of before the graphics exporter instance is closed or given a new source. All of the get and set functions for these sources are implemented by the base graphics exporter; format-specific importers should delegate all of them.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GraphicsExportGetInputImageDepth

Returns the depth of the input image for a graphics export operation.

```
ComponentResult GraphicsExportGetInputImageDepth (
   GraphicsExportComponent ci,
   long *inputDepth
);
```

Parameters

`ci`
The component instance that identifies your connection to the graphics exporter component.

`inputDepth`
Points to a variable to receive the input image depth.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GraphicsExportGetInputImageDescription

Returns an image description describing the input image in a graphics export operation.

```
ComponentResult GraphicsExportGetInputImageDescription (
   GraphicsExportComponent ci,
   ImageDescriptionHandle *desc
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

desc

Points to a variable to receive a handle to an `ImageDescription` structure.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function returns an `ImageDescription` structure containing information such as the format of the compressed data, its bit depth, natural bounds, and resolution.

Special Considerations

The caller is responsible for disposing of the returned image description handle.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

ElectricImageComponent

ElectricImageComponent.win

Declared In

`ImageCompression.h`

GraphicsExportGetInputImageDimensions

Returns the dimensions of the input image in a graphics export operation.

```
ComponentResult GraphicsExportGetInputImageDimensions (
   GraphicsExportComponent ci,
   Rect *dimensions
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

dimensions

Points to a rectangle to receive the dimensions of the input image.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
ElectricImageComponent

ElectricImageComponent.win

Declared In
`ImageCompression.h`

GraphicsExportGetInputOffsetAndLimit

Retrieves the current input offset and limit in a graphics export operation.

```
ComponentResult GraphicsExportGetInputOffsetAndLimit (
   GraphicsExportComponent ci,
   unsigned long *offset,
   unsigned long *limit
);
```

Parameters

`ci`
: The component instance that identifies your connection to the graphics exporter component.

`offset`
: Points to a variable to receive the offset. If you don't need this information, pass `NIL`.

`limit`
: Points to a variable to receive the limit. If you don't need this information, pass `NIL`.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function is only applicable when the input is a data reference, file, handle or pointer.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GraphicsExportGetInputPicture

Returns the current input picture in a graphics export operation.

```
ComponentResult GraphicsExportGetInputPicture (
   GraphicsExportComponent ci,
   PicHandle *picture
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

picture

Points to a variable to receive the source picture.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

You can use this function to get the source of a graphics export operation. The source can be a QuickTime graphics importer component instance, a QuickDraw `Picture`, a graphics world, a `PixMap` structure, or a piece of compressed data described by an `ImageDescription` structure. Compressed data can be in a file, handle, pointer, or other data reference. The application must make sure that the source is not disposed of before the graphics exporter instance is closed or given a new source. All of the get and set functions for these sources are implemented by the base graphics exporter; format-specific importers should delegate all of them.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GraphicsExportGetInputPixmap

Returns the current input pixmap in a graphics export operation.

```
ComponentResult GraphicsExportGetInputPixmap (
   GraphicsExportComponent ci,
   PixMapHandle *pixmap
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

pixmap

Points to a variable to receive the source `PixMap` structure.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

You can use this function to get the source of a graphics export operation. The source can be a QuickTime graphics importer component instance, a QuickDraw `Picture`, a graphics world, a `PixMap` structure, or a piece of compressed data described by an `ImageDescription` structure. Compressed data can be in a file, handle, pointer, or other data reference. The application must make sure that the source is not disposed of before the graphics exporter instance is closed or given a new source. All of the get and set functions for these sources are implemented by the base graphics exporter; format-specific importers should delegate all of them.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GraphicsExportGetInputPtr

Returns the current input pointer in a graphics export operation.

```
ComponentResult GraphicsExportGetInputPtr (
   GraphicsExportComponent ci,
   Ptr *p,
   unsigned long *size
);
```

Parameters

ci
: The component instance that identifies your connection to the graphics exporter component.

p
: A pointer to receive a pointer containing the graphics data.

size
: A pointer to a value describing the size of the image data in bytes.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
You can use this function to get the source of a graphics export operation. The source can be a QuickTime graphics importer component instance, a QuickDraw `Picture`, a graphics world, a `PixMap` structure, or a piece of compressed data described by an `ImageDescription` structure. Compressed data can be in a file, handle, pointer, or other data reference. The application must make sure that the source is not disposed of before the graphics exporter instance is closed or given a new source. All of the get and set functions for these sources are implemented by the base graphics exporter; format-specific importers should delegate all of them.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GraphicsExportGetInterlaceStyle

Returns the interlace style in a graphics export operation.

```
ComponentResult GraphicsExportGetInterlaceStyle (
   GraphicsExportComponent ci,
   unsigned long *interlaceStyle
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

interlaceStyle

Points to a variable to receive the interlace `style`. Valid values and interpretations are defined by individual exporters. In QuickTime 4, the PNG graphics exporter supports the `interlaceStyle` settings shown below See these constants:

`kQTPNGInterlaceNone`

`kQTPNGInterlaceAdam7`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GraphicsExportGetMetaData

Returns the current user data setting in a graphics export operation.

```
ComponentResult GraphicsExportGetMetaData (
   GraphicsExportComponent ci,
   void *userData
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

userData

A pointer to a `UserDataRecord` structure.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4. In QuickTime 4, none of the supplied graphics exporters support setting user data.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GraphicsExportGetMIMETypeList

Returns MIME types and other information about the graphics format in a graphics export operation.

```
ComponentResult GraphicsExportGetMIMETypeList (
   GraphicsExportComponent ci,
   void *qtAtomContainerPtr
);
```

Parameters

`ci`

The component instance that identifies your connection to the graphics exporter component.

`qtAtomContainerPtr`

Receives a newly-created QuickTime atom container that contains information about the graphics format.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function creates and returns a QuickTime atom container that contains the format's name, as a string in an atom of type `'desc'` (`kMimeInfoDescriptionTag`), and optionally the MIME type as a string in an atom of type `'mime'[atom]` (`kMimeInfoMimeTypeTag`).

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
ThreadsExporter

Declared In
`ImageCompression.h`

GraphicsExportGetOutputDataReference

Gets the output data reference handle in a graphics export operation.

```
ComponentResult GraphicsExportGetOutputDataReference (
   GraphicsExportComponent ci,
   Handle *dataRef,
   OSType *dataRefType
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

dataRef

Points to a variable to receive the data reference handle.

dataRefType

Points to a variable to receive a constant that identifies the `data` reference type. See `Data References`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Special Considerations

The caller is responsible for disposing of the returned data reference handle.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GraphicsExportGetOutputFile

Returns the current output file for a graphics export operation.

```
ComponentResult GraphicsExportGetOutputFile (
   GraphicsExportComponent ci,
   FSSpec *theFile
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

theFile

Points to a variable to receive the `FSSpec`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GraphicsExportGetOutputFileTypeAndCreator

Gets the type and creator codes for the output file in a graphics export operation.

```
ComponentResult GraphicsExportGetOutputFileTypeAndCreator (
   GraphicsExportComponent ci,
   OSType *fileType,
   OSType *fileCreator
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

fileType

Receives the file type for the new image file. See `File Types and Creators`.

fileCreator

Receives the file creator for the new image file. See `File Types and Creators`.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GraphicsExportGetOutputHandle

Returns the current output handle for a graphics export operation.

```
ComponentResult GraphicsExportGetOutputHandle (
   GraphicsExportComponent ci,
   Handle *h
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

h

Points to a variable to receive the handle.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GraphicsExportGetOutputMark

Returns the current file position for a graphics export operation.

```
ComponentResult GraphicsExportGetOutputMark (
   GraphicsExportComponent ci,
   unsigned long *mark
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

mark

Receives the current file position, as a byte offset from the beginning of the output data reference.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Special Considerations

Not all output data types support the current file position feature.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GraphicsExportGetOutputOffsetAndMaxSize

Returns the output starting offset and maximum size limit for a graphics export operation.

```
ComponentResult GraphicsExportGetOutputOffsetAndMaxSize (
   GraphicsExportComponent ci,
   unsigned long *offset,
   unsigned long *maxSize,
   Boolean *truncateFile
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

offset

On return, a value describing the byte offset of the image data from the beginning of the data reference. If you are not interested in this information, you may pass `NIL`.

maxSize

On return, a value describing the maximum size limit. If you are not interested in this information, you may pass `NIL`.

truncateFile

A Boolean value; TRUE means to truncate the file, FALSE means not.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GraphicsExportGetProgressProc

Returns the current progress function for a graphics export operation.

```
ComponentResult GraphicsExportGetProgressProc (
   GraphicsExportComponent ci,
   ICMProgressProcRecordPtr progressProc
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

progressProc

A pointer to an `ICMProgressProc` callback.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
By default, graphics export components have no progress functions.

Special Considerations

This function is always implemented by the base graphics exporter.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GraphicsExportGetResolution

Determines the resolution of a graphics exporter component.

```
ComponentResult GraphicsExportGetResolution (
   GraphicsExportComponent ci,
   Fixed *horizontalResolution,
   Fixed *verticalResolution
);
```

Parameters

ci

A component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

horizontalResolution

Points to a variable to receive the horizontal resolution.

verticalResolution

Points to a variable to receive the vertical resolution.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GraphicsExportGetSettingsAsAtomContainer

Retrieves the current settings from a graphics exporter component.

```
ComponentResult GraphicsExportGetSettingsAsAtomContainer (
   GraphicsExportComponent ci,
   void *qtAtomContainerPtr
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

qtAtomContainerPtr

Points to a variable to receive a new QuickTime atom container containing the current graphics exporter component settings.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Special Considerations

The caller is responsible for disposing of the returned atom container.

Version Notes

Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Fiendishthngs

Declared In
`ImageCompression.h`

GraphicsExportGetSettingsAsText

Retrieves the current settings from the graphics export component in a user-readable format.

```
ComponentResult GraphicsExportGetSettingsAsText (
   GraphicsExportComponent ci,
   Handle *theText
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

theText

Points to a variable to receive a newly-allocated handle containing text.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Special Considerations

The caller is responsible for disposing of the returned handle.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GraphicsExportGetTargetDataSize

Returns the current desired maximum data size for a graphics export operation.

```
ComponentResult GraphicsExportGetTargetDataSize (
   GraphicsExportComponent ci,
   unsigned long *targetDataSize
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

targetDataSize

Points to a variable to receive the desired maximum data size in bytes.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GraphicsExportGetThumbnailEnabled

Returns the current thumbnail creation setting for the graphics exporter when exporting Exif files.

```
ComponentResult GraphicsExportGetThumbnailEnabled (
   GraphicsExportComponent ci,
   Boolean *thumbnailEnabled,
   long *maxThumbnailWidth,
   long *maxThumbnailHeight
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component. This function is supported only by the TIFF and JPEG graphics exporters.

thumbnailEnabled

Points to a variable to receive the current thumbnail setting. Pass `NIL` if you do not want to receive this information.

maxThumbnailWidth

Points to a variable to receive the current maximum thumbnail width. Pass `NIL` if you do not want to receive this information.

maxThumbnailHeight

Points to a variable to receive the current maximum thumbnail height. Pass `NIL` if you do not want to receive this information.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 6.

Availability

Available in Mac OS X v10.1 and later.

Declared In

`ImageCompression.h`

GraphicsExportMayExporterReadInputData

Asks whether the image source for a graphics export operation is in a form that the exporter can read.

```
ComponentResult GraphicsExportMayExporterReadInputData (
   GraphicsExportComponent ci,
   Boolean *mayReadInputData
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

mayReadInputData

Points to a Boolean; TRUE means that the image source is in a form that the exporter can read, FALSE means it is not.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Some kinds of image source, such as files and handles, form a stream of bytes that can be read directly. Others, such as pictures and pixmaps, do not. Format-specific graphics exporters usually cannot transcode if they cannot read the original data, so those exporters which implement `GraphicsExportCanTranscode` (page 974) will usually first call `GraphicsExportMayExporterReadInputData`.

Special Considerations

This function is used by format-specific graphics exporters when transcoding. Applications will not normally need to call this function.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GraphicsExportReadInputData

Reads the original image data in a graphics export operation.

```
ComponentResult GraphicsExportReadInputData (
   GraphicsExportComponent ci,
   void *dataPtr,
   unsigned long dataOffset,
   unsigned long dataSize
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

dataPtr

A pointer to a memory block to receive the data.

dataOffset
: The offset of the image data within the source image data. The function begins reading image data from this offset.

dataSize
: The number of bytes of image data to read.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function communicates with the appropriate data handler to retrieve image data.

Special Considerations

This function is used by format-specific graphics exporters when transcoding. Applications will not normally need to call this function.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GraphicsExportReadOutputData

Reads output image data in a graphics export operation.

```
ComponentResult GraphicsExportReadOutputData (
   GraphicsExportComponent ci,
   void *dataPtr,
   unsigned long dataOffset,
   unsigned long dataSize
);
```

Parameters

ci
: The component instance that identifies your connection to the graphics exporter component.

dataPtr
: A pointer to a memory block to receive the data.

dataOffset
: The offset of the image data within the data reference. The function begins reading image data from this offset.

dataSize
: The number of bytes of image data to read.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Special Considerations

Not all output data types support this function.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GraphicsExportRequestSettings

Displays a dialog for the user to configure graphics exporter settings, if applicable.

```
ComponentResult GraphicsExportRequestSettings (
   GraphicsExportComponent ci,
   ModalFilterYDUPP filterProc,
   void *yourDataProc
);
```

Parameters

ci
: The component instance that identifies your connection to the graphics exporter component.

filterProc
: A `ModalFilterYDProc` callback. If you don't need one, pass `NIL`.

yourDataProc
: An extra parameter that will be passed to the `ModalFilterProc` callback when it is called. If you don't need one, pass `NIL`.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Some graphics exporters don't support settings dialogs, and so don't implement this call. To find out whether a graphics exporter implements this call, you can use this code:

```
CallComponentCanDo( myGraphicsExporter,
                    kGraphicsExportRequestSettingsSelect);
```

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Graphic Import-Export
ImproveYourImage

Declared In
`ImageCompression.h`

GraphicsExportSetColorSyncProfile

Sets the ColorSync profile to embed in the image file for a graphics export operation.

```
ComponentResult GraphicsExportSetColorSyncProfile (
   GraphicsExportComponent ci,
   Handle colorSyncProfile
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

colorSyncProfile

A handle to the ColorSync profile.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

ColorSync profiles allow image files to describe their native colorspace in a self-contained manner. They can be stored in atoms of type `'iicc'`.

Version Notes

Introduced in QuickTime 4. Starting with QuickTime 4, the JPEG, PNG, PICT, QuickTime Image and TIFF graphics exporters support embedded ColorSync profiles.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GraphicsExportSetCompressionMethod

Defines the compression method to use in a graphics export operation.

```
ComponentResult GraphicsExportSetCompressionMethod (
   GraphicsExportComponent ci,
   long compressionMethod
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

compressionMethod

A value (see below) describing the compression algorithm to be used by the graphics exporter. See these constants:

`kQTTIFFCompression_None`

`kQTTIFFCompression_PackBits`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

In QuickTime 4, the TIFF graphics exporter supports the `compressionMethod` settings `kQTTIFFCompression_None` and `kQTTIFFCompression_PackBits`. Some image formats, such as TIFF, support several compression methods.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GraphicsExportSetCompressionQuality

Defines the compression quality for a graphics export operation.

```
ComponentResult GraphicsExportSetCompressionQuality (
   GraphicsExportComponent ci,
   CodecQ spatialQuality
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

spatialQuality

A constant (see below) that defines the currently specified quality value. See these constants:

`codecMinQuality`

`codecLowQuality`

`codecNormalQuality`

`codecHighQuality`

`codecMaxQuality`

`codecLosslessQuality`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This setting is only supported by lossy compression methods.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Carbon GLSnapshot

qtgraphics

qtgraphics.win

TextNameTool

ThreadsExporter

Declared In
`ImageCompression.h`

GraphicsExportSetDepth

Defines the depth to use in a graphics export operation.

```
ComponentResult GraphicsExportSetDepth (
   GraphicsExportComponent ci,
   long depth
);
```

Parameters

`ci`
: The component instance that identifies your connection to the graphics exporter component.

`depth`
: A value describing the depth of the image data. Some image file formats support more than one pixel depth.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
The BMP, JPEG, Photoshop, PNG, PICT, QuickTime Image, TGA and TIFF graphics exporters support the depth setting. Some image file formats support more than one pixel depth.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Graphic Import-Export
ImproveYourImage
TextNameTool

Declared In
`ImageCompression.h`

GraphicsExportSetDontRecompress

Requests that the original compressed data for a graphics export operation not be decompressed and recompressed, but be copied through to the output file.

```
ComponentResult GraphicsExportSetDontRecompress (
   GraphicsExportComponent ci,
   Boolean dontRecompress
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

dontRecompress

If TRUE, requests not to recompress the image data.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Even though it is not decompressed and recompressed, graphics data may be modified when it is copied through.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GraphicsExportSetExifEnabled

Determines whether or not the graphics exporter component should create Exif files.

```
ComponentResult GraphicsExportSetExifEnabled (
   GraphicsExportComponent ci,
   Boolean enableExif
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component. This function is supported only by the TIFF and JPEG graphics exporters.

enableExif

Pass TRUE to enable `Exif` file creation.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Turning on `Exif` export disables incompatible settings, such as grayscale JPEG and compressed TIFF, and enables export of `Exif` metadata.

Version Notes

Introduced in QuickTime 6.

Availability

Available in Mac OS X v10.1 and later.

Declared In
`ImageCompression.h`

GraphicsExportSetInputCGBitmapContext

Sets the CGBitmapContext that the graphics exporter will use as its input image.

```
ComponentResult GraphicsExportSetInputCGBitmapContext (
   GraphicsExportComponent ci,
   CGContextRef bitmapContextRef
);
```

Parameters

ci
: The component instance that identifies your connection to the graphics exporter component.

bitmapContextRef
: A reference to the Core Graphics context.

Return Value
See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 6.4.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`ImageCompression.h`

GraphicsExportSetInputCGImage

Specifies a Core Graphics CGImage as the source for a graphics export operation.

```
ComponentResult GraphicsExportSetInputCGImage (
   GraphicsExportComponent ci,
   CGImageRef imageRef
);
```

Parameters

ci
: The component instance that identifies your connection to the graphics exporter component.

imageRef
: A reference to a `CG` image.

Return Value
See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 6.4.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`ImageCompression.h`

GraphicsExportSetInputDataReference

Specifies that the source image for a graphics export operation is a compressed image stored in a data reference.

```
ComponentResult GraphicsExportSetInputDataReference (
   GraphicsExportComponent ci,
   Handle dataRef,
   OSType dataRefType,
   ImageDescriptionHandle desc
);
```

Parameters
`ci`
: The component instance that identifies your connection to the graphics exporter component.

`dataRef`
: A QuickTime data reference. See `Data References`.

`dataRefType`
: The type of the data reference; see `Data References`.

`desc`
: A handle to an `ImageDescription` structure, describing the compressed data.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
You can use this function to specify a source before you call `GraphicsExportDoExport` (page 975). The source can be a QuickTime graphics importer component instance, a QuickDraw `Picture`, a graphics world, a `PixMap` structure, or a piece of compressed data described by an `ImageDescription` structure. Compressed data can be in a file, handle, pointer, or other data reference. The application must make sure that the source is not disposed of before the graphics exporter instance is closed or given a new source. All of the get and set functions for these sources are implemented by the base graphics exporter; format-specific importers should delegate all of them.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GraphicsExportSetInputFile

Specifies that the source image for a graphics export operation is a compressed image stored in a file.

```
ComponentResult GraphicsExportSetInputFile (
   GraphicsExportComponent ci,
   const FSSpec *theFile,
   ImageDescriptionHandle desc
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

theFile

A pointer to the `FSSpec` structure for the file containing the graphics data.

desc

A handle to an `ImageDescription` structure that describes the compressed data.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

You can use this function to specify a source before you call `GraphicsExportDoExport` (page 975). The source can be a QuickTime graphics importer component instance, a QuickDraw `Picture`, a graphics world, a `PixMap` structure, or a piece of compressed data described by an `ImageDescription` structure. Compressed data can be in a file, handle, pointer, or other data reference. The application must make sure that the source is not disposed of before the graphics exporter instance is closed or given a new source. All of the get and set functions for these sources are implemented by the base graphics exporter; format-specific importers should delegate all of them.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GraphicsExportSetInputGraphicsImporter

Specifies that the source image for a graphics export operation is to be drawn by a graphics importer instance.

```
ComponentResult GraphicsExportSetInputGraphicsImporter (
   GraphicsExportComponent ci,
   GraphicsImportComponent grip
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

grip

The source graphics importer component instance.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

You can use this function to specify a source before you call `GraphicsExportDoExport` (page 975). The source can be a QuickTime graphics importer component instance, a QuickDraw `Picture`, a graphics world, a `PixMap` structure, or a piece of compressed data described by an `ImageDescription` structure. Compressed data can be in a file, handle, pointer, or other data reference. The application must make sure that the source is not disposed of before the graphics exporter instance is closed or given a new source. All of the get and set functions for these sources are implemented by the base graphics exporter; format-specific importers should delegate all of them.

Special Considerations

It is the caller's responsibility to dispose of the graphics importer.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qtgraphics

qtgraphics.win

ThreadsExporter

Declared In

`ImageCompression.h`

GraphicsExportSetInputGWorld

Specifies that the source image for a graphics export operation is a graphics world.

```
ComponentResult GraphicsExportSetInputGWorld (
   GraphicsExportComponent ci,
   GWorldPtr gworld
);
```

Parameters

`ci`

The component instance that identifies your connection to the graphics exporter component.

`gworld`

The source graphics world. It must be a real graphics world; you may not pass an ordinary color `GrafPort`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

You can use this function to specify a source before you call `GraphicsExportDoExport` (page 975). The source can be a QuickTime graphics importer component instance, a QuickDraw `Picture`, a graphics world, a `PixMap` structure, or a piece of compressed data described by an `ImageDescription` structure. Compressed data can be in a file, handle, pointer, or other data reference. The application must make sure that the source is not disposed of before the graphics exporter instance is closed or given a new source. All of the get and set functions for these sources are implemented by the base graphics exporter; format-specific importers should delegate all of them.

Special Considerations

The graphics exporter will never dispose the graphics world.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Carbon GLSnapshot

Graphic Import-Export

ImproveYourImage

TextNameTool

Declared In

`ImageCompression.h`

GraphicsExportSetInputHandle

Specifies that the source image for a graphics export operation is a compressed image referenced by a handle.

```
ComponentResult GraphicsExportSetInputHandle (
   GraphicsExportComponent ci,
   Handle h,
   ImageDescriptionHandle desc
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

h

A handle to graphics data.

desc

A handle to an `ImageDescription` structure that describes the compressed data.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

You can use this function to specify a source before you call `GraphicsExportDoExport` (page 975). The source can be a QuickTime graphics importer component instance, a QuickDraw `Picture`, a graphics world, a `PixMap` structure, or a piece of compressed data described by an `ImageDescription` structure. Compressed data can be in a file, handle, pointer, or other data reference. The application must make sure that the source is not disposed of before the graphics exporter instance is closed or given a new source. All of the get and set functions for these sources are implemented by the base graphics exporter; format-specific importers should delegate all of them.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

ImageCompression.h

GraphicsExportSetInputOffsetAndLimit

Specifies the portion of an input data reference, file, handle or pointer that a graphics exporter is permitted to read.

```
ComponentResult GraphicsExportSetInputOffsetAndLimit (
   GraphicsExportComponent ci,
   unsigned long offset,
   unsigned long limit
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

offset

The byte offset of the input image data from the beginning of the data reference.

limit

The offset of the byte following the last byte of the input image data. (If you don't need to apply any limit, pass (unsigned long)-1.) Both the offset parameter and the limit parameter values are relative to the start of the compressed data. GraphicsExportGetInputDataSize (page 986) and GraphicsExportReadInputData (page 1003) take the offset and limit values into account automatically.

Return Value

See Error Codes. Returns noErr if there is no error.

Discussion

This routine would be useful if, for example, the source was a JPEG image embedded within a larger file.

Special Considerations

This function is only applicable when the input is a data reference, file, handle, or pointer.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

ImageCompression.h

GraphicsExportSetInputPicture

Specifies that the source image for a graphics export operation is a picture.

```
ComponentResult GraphicsExportSetInputPicture (
   GraphicsExportComponent ci,
   PicHandle picture
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

picture

A handle to the source picture.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

You can use this function to specify a source before you call `GraphicsExportDoExport` (page 975). The source can be a QuickTime graphics importer component instance, a QuickDraw `Picture`, a graphics world, a `PixMap` structure, or a piece of compressed data described by an `ImageDescription` structure. Compressed data can be in a file, handle, pointer, or other data reference. The application must make sure that the source is not disposed of before the graphics exporter instance is closed or given a new source. All of the get and set functions for these sources are implemented by the base graphics exporter; format-specific importers should delegate all of them.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GraphicsExportSetInputPixmap

Specifies that the source image for a graphics export operation is a pixmap.

```
ComponentResult GraphicsExportSetInputPixmap (
   GraphicsExportComponent ci,
   PixMapHandle pixmap
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

pixmap

The source `PixMap` structure.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

You can use this function to specify a source before you call `GraphicsExportDoExport` (page 975). The source can be a QuickTime graphics importer component instance, a QuickDraw `Picture`, a graphics world, a `PixMap` structure, or a piece of compressed data described by an `ImageDescription` structure. Compressed data can be in a file, handle, pointer, or other data reference. The application must make sure that the source is not disposed of before the graphics exporter instance is closed or given a new source. All of the get and set functions for these sources are implemented by the base graphics exporter; format-specific importers should delegate all of them.

Special Considerations

It is the caller's responsibility to dispose of the `pixmap`.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GraphicsExportSetInputPtr

Specifies that the source image for a graphics export operation is a compressed image stored at a fixed address in memory.

```
ComponentResult GraphicsExportSetInputPtr (
   GraphicsExportComponent ci,
   Ptr p,
   unsigned long size,
   ImageDescriptionHandle desc
);
```

Parameters

`ci`

The component instance that identifies your connection to the graphics exporter component.

`p`

A pointer to a value the image.

`size`

A value describing the size of the image data in bytes.

`desc`

A handle to an `ImageDescription` structure that describes the compressed data.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

You can use this function to specify a source before you call `GraphicsExportDoExport` (page 975). The source can be a QuickTime graphics importer component instance, a QuickDraw `Picture`, a graphics world, a `PixMap` structure, or a piece of compressed data described by an `ImageDescription` structure. Compressed data can be in a file, handle, pointer, or other data reference. The application must make sure that the source is not disposed of before the graphics exporter instance is closed or given a new source. All of the get and set functions for these sources are implemented by the base graphics exporter; format-specific importers should delegate all of them.

Version Notes

Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GraphicsExportSetInterlaceStyle

Defines the interlace style for a graphics export operation.

```
ComponentResult GraphicsExportSetInterlaceStyle (
   GraphicsExportComponent ci,
   unsigned long interlaceStyle
);
```

Parameters

`ci`
The component instance that identifies your connection to the graphics exporter component.

`interlaceStyle`
The new interlace `style` to use. Valid values and interpretations are defined by individual exporters. In QuickTime 4, the PNG graphics exporter supports the `interlaceStyle` settings shown below.
See these constants:
`kQTPNGInterlaceNone`
`kQTPNGInterlaceAdam7`

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
A common use for this function is in the PNG and GIF formats, which rearrange data so that low-resolution images can be displayed from incomplete data streams.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GraphicsExportSetMetaData

Defines supplemental data for a graphics export operation, such as copyright text.

```
ComponentResult GraphicsExportSetMetaData (
   GraphicsExportComponent ci,
   void *userData
);
```

Parameters

ci
: The component instance that identifies your connection to the graphics exporter component.

userData
: A pointer to user data. The value you pass should have the type userData, which is a pointer to a `UserDataRecord`.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Special Considerations

In QuickTime 4, none of the supplied graphics exporters support setting user data.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GraphicsExportSetOutputDataReference

Returns the current output data reference for a graphics export operation.

```
ComponentResult GraphicsExportSetOutputDataReference (
   GraphicsExportComponent ci,
   Handle dataRef,
   OSType dataRefType
);
```

Parameters

ci
: The component instance that identifies your connection to the graphics exporter component.

dataRef
: A QuickTime data reference.

dataRefType
: The type of the data reference; see `Data References`.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GraphicsExportSetOutputFile

Defines the output file for a graphics export operation.

```
ComponentResult GraphicsExportSetOutputFile (
   GraphicsExportComponent ci,
   const FSSpec *theFile
);
```

Parameters

`ci`
The component instance that identifies your connection to the graphics exporter component.

`theFile`
an `FSSpec` structure that identifies the file.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Graphic Import-Export
ImproveYourImage
qtgraphics
qtgraphics.win
ThreadsExporter

Declared In
`ImageCompression.h`

GraphicsExportSetOutputFileTypeAndCreator

Sets the file type and creator codes for the output file of a graphics export operation.

```
ComponentResult GraphicsExportSetOutputFileTypeAndCreator (
   GraphicsExportComponent ci,
   OSType fileType,
   OSType fileCreator
);
```

Parameters

`ci`
The component instance that identifies your connection to the graphics exporter component.

`fileType`
The file type for the new image file, such as `'JPEG'`. See `File Types and Creators`.

fileCreator

The file creator for the new image file. This parameter may be 0, in which case a default file creator for this file type is used. See `File Types and Creators`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GraphicsExportSetOutputHandle

Sets a handle to the output of a graphics export operation.

```
ComponentResult GraphicsExportSetOutputHandle (
   GraphicsExportComponent ci,
   Handle h
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

h

The output handle.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

TextNameTool

Declared In

`ImageCompression.h`

GraphicsExportSetOutputMark

Seeks to the specified file position in a graphics export operation.

```
ComponentResult GraphicsExportSetOutputMark (
   GraphicsExportComponent ci,
   unsigned long mark
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

mark

The new file position, specified as a byte offset from the beginning of the output data reference.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GraphicsExportSetOutputOffsetAndMaxSize

Specifies the output starting offset and maximum size limit for a graphics export operation.

```
ComponentResult GraphicsExportSetOutputOffsetAndMaxSize (
   GraphicsExportComponent ci,
   unsigned long offset,
   unsigned long maxSize,
   Boolean truncateFile
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

offset

The byte offset of the image data from the beginning of the data reference.

maxSize

A value describing the maximum size limit.

truncateFile

A Boolean value; TRUE means to truncate the file.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GraphicsExportSetProgressProc

Installs a progress function in a graphics export operation.

```
ComponentResult GraphicsExportSetProgressProc (
   GraphicsExportComponent ci,
   ICMProgressProcRecordPtr progressProc
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

progressProc

Points to an `ICMProgressProc` callback. If you pass a value of -1, QuickTime provides a standard progress function. If you want to remove the existing progress function, pass `NIL`.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function is always implemented by the base graphics exporter.

Special Considerations

If your progress function does any drawing, you should take care to set a safe graphics state before doing so, and to restore the graphics state afterwards. In particular, the current graphics device may be an offscreen device.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
ThreadsExporter

Declared In
`ImageCompression.h`

GraphicsExportSetResolution

Defines the resolution to store in the image file for a graphics export operation.

```
ComponentResult GraphicsExportSetResolution (
   GraphicsExportComponent ci,
   Fixed horizontalResolution,
   Fixed verticalResolution
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

horizontalResolution

A value describing the horizontal resolution of the image, where the upper byte is dots per inch. The value 0x00480000 represents 72.0 dpi.

verticalResolution

A value describing the vertical resolution of the image, where the upper byte is dots per inch. The value 0x00480000 represents 72.0 dpi.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GraphicsExportSetSettingsFromAtomContainer

Sets the graphics exporter component's current configuration to match the settings in a passed atom container.

```
ComponentResult GraphicsExportSetSettingsFromAtomContainer (
   GraphicsExportComponent ci,
   void *qtAtomContainer
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

qtAtomContainer

A pointer to a QuickTime atom container that contains settings.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The settings atom container may contain atoms other than those expected by the graphics exporter component or may be missing certain atoms. This function will use only the settings it understands.

Version Notes

Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GraphicsExportSetTargetDataSize

Defines a desired maximum data size for a graphics export operation and asks for a quality that does not exceed that size.

```
ComponentResult GraphicsExportSetTargetDataSize (
   GraphicsExportComponent ci,
   unsigned long targetDataSize
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

targetDataSize

A value that describes the maximum size of the image data in bytes.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GraphicsExportSetThumbnailEnabled

Determines whether or not the graphics exporter component should create an embedded thumbnail inside an exported Exif file.

```
ComponentResult GraphicsExportSetThumbnailEnabled (
   GraphicsExportComponent ci,
   Boolean enableThumbnail,
   long maxThumbnailWidth,
   long maxThumbnailHeight
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component. This function is currently supported only by the TIFF and JPEG graphics exporters.

enableThumbnail

Pass TRUE to turn thumbnail creation on; otherwise pass FALSE.

maxThumbnailWidth

The maximum width for created thumbnails.

maxThumbnailHeight

The maximum height for created thumbnails. If one maximum dimension is 0, only the other will be used. If both maximum dimensions are 0, the graphics exporter will decide for itself. The graphics exporter will not change the aspect ratio of the `Exif` image when creating the thumbnail, nor will it create a thumbnail larger than the image.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The JPEG graphics exporter can create thumbnails only when writing `Exif` files.

Version Notes

Introduced in QuickTime 6.

Availability

Available in Mac OS X v10.1 and later.

Declared In

`ImageCompression.h`

GraphicsExportWriteOutputData

Writes output image data in a graphics export operation.

```
ComponentResult GraphicsExportWriteOutputData (
   GraphicsExportComponent ci,
   const void *dataPtr,
   unsigned long dataSize
);
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

dataPtr

A pointer to a memory block containing the data.

dataSize

The number of bytes of image data to write.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function is used by format-specific graphics exporters to write output data.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code
ElectricImageComponent

ElectricImageComponent.win

Declared In
`ImageCompression.h`

GraphicsImportCreateCGImage

Imports an image as a Core Graphics CGImage.

```
ComponentResult GraphicsImportCreateCGImage (
   GraphicsImportComponent ci,
   CGImageRef *imageRefOut,
   UInt32 flags
);
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

imageRefOut

A reference to the `CG` image to be created.

flags

A flag (see below) that determines the settings to use.
`kGraphicsImportCreateCGImageUsingCurrentSettings` Use the current settings. See these constants:

`kGraphicsImportCreateCGImageUsingCurrentSettings`

Return Value
See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 6.4.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`ImageCompression.h`

GraphicsImportDoesDrawAllPixels

Asks whether the graphics importer expects to draw every pixel.

```
ComponentResult GraphicsImportDoesDrawAllPixels (
   GraphicsImportComponent ci,
   short *drawsAllPixels
);
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

drawsAllPixels

A pointer to a value (see below) that describes the predicted drawing behavior. See these constants:

`graphicsImporterDrawsAllPixels`

`graphicsImporterDoesntDrawAllPixels`

`graphicsImporterDontKnowIfDrawAllPixels`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Some image file formats permit non-rectangular images or images with transparent regions. When such an image is drawn, not every pixel in the boundary rectangle will be changed. `GraphicsImportDoesDrawAllPixels` lets you try to find out whether this will be the case. For instance, you might choose to erase the area behind the image before drawing. If the graphics import component supports this function, `drawsAllPixels` will contain one of the constants shown above on return.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Graphic Import-Export

ImproveYourImage

Declared In

`ImageCompression.h`

GraphicsImportDoExportImageFileDialog

Presents a dialog box letting the user save an imported image in a foreign file format.

```
ComponentResult GraphicsImportDoExportImageFileDialog (
   GraphicsImportComponent ci,
   const FSSpec *inDefaultSpec,
   StringPtr prompt,
   ModalFilterYDUPP filterProc,
   OSType *outExportedType,
   FSSpec *outExportedSpec,
   ScriptCode *outScriptTag
);
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

inDefaultSpec

A pointer to an `FSSpec` that suggests a default name for the file. If you don't want to suggest a default name, pass `NIL`.

prompt

A pointer to a prompt string that appears in the standard put dialog box; it may be `NIL`, in which case a default string is used.

filterProc
: A modal filter function to be passed to the Mac OS function `CustomPutFile`; see *Inside Macintosh: Files* for more information. If you don't need to filter events, pass `NIL`.

outExportedType
: A pointer to a variable that will receive the type of the export file that was chosen by the user. If you don't want this information, pass `NIL`. See `File Types and Creators`.

outExportedSpec
: A pointer to a variable that will receive the `FSSpec` of the file that was written. If you don't want this information, pass `NIL`.

outScriptTag
: A pointer to a variable that will receive the script system in which the exported file name is to be displayed. See `Localization Codes`. If you don't want this information, pass `NIL`.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function presents the user with an extended Standard File dialog box that allows the image currently in use by the graphics import component to be exported to a file, in a format of the user's choice.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qtgraphics
qtgraphics.win
QTGraphicsImport
qtgraphimp
qtgraphimp.win

Declared In
`ImageCompression.h`

GraphicsImportDoExportImageFileToDataRefDialog

Presents a dialog box that lets the user save an imported image in a foreign file format.

```
ComponentResult GraphicsImportDoExportImageFileToDataRefDialog (
   GraphicsImportComponent ci,
   Handle inDataRef,
   OSType inDataRefType,
   CFStringRef prompt,
   ModalFilterYDUPP filterProc,
   OSType *outExportedType,
   Handle *outDataRef,
   OSType *outDataRefType
);
```

Parameters

ci
: The component instance that identifies your connection to the graphics importer component.

inDefaultDataRef
: A data reference that specifies the default export location.

inDefaultDataRefType
: The type of the data reference that specifies the default export location.

prompt
: A reference to a `CFString` that contains the prompt text string for the dialog.

filterProc
: A modal filter function; see `ModalFilterYDProc` in the QuickTime API Reference.

outExportedType
: A pointer to an `OSType` entity where the type of the exported file will be returned.

outExportedDataRef
: A pointer to an handle where the data reference to the exported file will be returned.

outExportedDataRefType
: A pointer to an `OSType` entity where the type of the data reference that points to the exported file will be returned.

Return Value
See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error.

Discussion
This function presents a file dialog that lets the user to specify a file to which the exported data goes and a format into which image data is exported. By using data references, a long file name or Unicode file name can be used as a default file name as well as the name of the file into which the export data goes. This function is equivalent to `GraphicsImportDoExportImageFileDialog`.

Version Notes
Introduced in QuickTime 6.4.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`ImageCompression.h`

GraphicsImportDraw

Draws an imported image.

```
ComponentResult GraphicsImportDraw (
   GraphicsImportComponent ci
);
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function draws the image currently in use by the graphics import component to the graphics port and device specified by `GraphicsImportSetGWorld` (page 1072). `GraphicsImportDraw` takes into account all settings previously specified for the image, such as the source rectangle, transformation matrix, clipping region, graphics mode, and image quality.

Special Considerations

The base graphics importer's drawing function uses the results of `GraphicsImportGetImageDescription` (page 1052) and `GraphicsImportGetDataOffsetAndSize` (page 1039) to create a decompression sequence, which it uses to draw the image. Subsequent draw operations with the same connection may reuse the decompression sequence. Other graphics importers may override this behavior.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Graphic Import-Export

ImproveYourImage

qtgraphics

qtgraphics.win

vrmakepano

Declared In

`ImageCompression.h`

GraphicsImportExportImageFile

Saves an imported image in a foreign file format.

```
ComponentResult GraphicsImportExportImageFile (
   GraphicsImportComponent ci,
   OSType fileType,
   OSType fileCreator,
   const FSSpec *fss,
   ScriptCode scriptTag
);
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

fileType

The file type for the new image file, such as 'JPEG'. See File Types and Creators.

fileCreator

The file creator for the new image file. See File Types and Creators. You may pass 0, in which case a default file creator for this file type is used.

fss

A pointer to the FSSpec structure that identifies the file that is to receive the exported image.

scriptTag

The script system in which the file name is to be displayed; see Localization Codes. If you have established the name and location of the file using one of the Standard File Package functions, use the script code returned in the reply record (reply.sfScript). Otherwise, specify the system script by setting the scriptTag parameter to the value smSystemScript. See *Inside Macintosh: Files* for more information about script specifications.

Return Value

See Error Codes. Returns noErr if there is no error.

Discussion

This function creates a new file containing the image currently in use by the graphics import component. The new file is compressed in a format corresponding to the provided file type. If a non-identity matrix has been applied to the graphics import component, this matrix is applied to the image before export. Since most image formats don't support nonzero top-left coordinates, the matrix is temporarily adjusted to ensure that the exported image's bounds have top-left coordinates at (0,0). If the matrix does not map the graphics import component's source rectangle to a rectangle, there will be extra white space left around the image.

Special Considerations

Graphics import components can save data in several formats, including QuickDraw pictures and QuickTime Image files. This capability is only needed by applications that perform file format translation. Applications that only wish to draw the image can use GraphicsImportDraw (page 1030).

Version Notes

In QuickTime 3, the supported export file types are kQTFileTypePicture, kQTFileTypeQuickTimeImage, kQTFileTypeBMP, kQTFileTypeJPEG, and kQTFileTypePhotoShop. QuickTime 4 uses graphics exporter components to implement image export.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qtgraphics

qtgraphics.win

qtgraphimp

qtgraphimp.win

Declared In
`ImageCompression.h`

GraphicsImportExportImageFileToDataRef

Saves an imported image in a foreign file format.

```
ComponentResult GraphicsImportExportImageFileToDataRef (
   GraphicsImportComponent ci,
   OSType fileType,
   OSType fileCreator,
   Handle dataRef,
   OSType dataRefType
);
```

Parameters

ci
The component instance that identifies your connection to the graphics importer component.

fileType
The Mac OS file type for the new file, which determines the file format.

fileCreator
The creator type of the new file.

dataRef
A data reference that specifies a storage location to which the image is to be exported.

dataRefType
The type of the data reference.

Return Value
See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error.

Discussion
This function exports the imported image as a foreign file format specified by `fileType`. The exported data will be saved into a storage location specified by a data reference. You can use data reference functions to create a data reference for a file that has long or Unicode file name. This function is equivalent to `GraphicsImportExportImageFile`.

Version Notes
Introduced in QuickTime 6.4.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`ImageCompression.h`

GraphicsImportGetAliasedDataReference

Deprecated.

```
ComponentResult GraphicsImportGetAliasedDataReference (
   GraphicsImportComponent ci,
   Handle *dataRef,
   OSType *dataRefType
);
```

Version Notes

This function is listed for historical purposes only. It may be unsupported or removed in future versions of QuickTime.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GraphicsImportGetAsPicture

Creates a QuickDraw picture handle to an imported image.

```
ComponentResult GraphicsImportGetAsPicture (
   GraphicsImportComponent ci,
   PicHandle *picture
);
```

Parameters

`ci`

The component instance that identifies your connection to the graphics importer component.

`picture`

Points to a handle to a `Picture` structure that is to receive the image.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function creates a new QuickDraw picture handle containing the image currently in use by the graphics import component. If possible, the image will remain in the compressed format. For example, if the image is from a JFIF file, the picture will contain compressed JPEG data. It is the responsibility of the caller to dispose of the picture handle.

Special Considerations

Graphics import components can save data in several formats, including QuickDraw pictures and QuickTime Image files. This capability is only needed by applications that perform file format translation. Applications that only wish to draw the image can use `GraphicsImportDraw` (page 1030).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Graphic Import-Export

ImproveYourImage

qtskins
qtskins.win
SampleDS

Declared In
`ImageCompression.h`

GraphicsImportGetBaseDataOffsetAndSize64

Undocumented

```
ComponentResult GraphicsImportGetBaseDataOffsetAndSize64 (
   GraphicsImportComponent ci,
   wide *offset,
   wide *size
);
```

Parameters

ci
The component instance that identifies your connection to the graphics importer component.

offset
Undocumented

size
Undocumented

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.1 and later.

Declared In
`ImageCompression.h`

GraphicsImportGetBoundsRect

Returns the bounding rectangle for drawing an imported image.

```
ComponentResult GraphicsImportGetBoundsRect (
   GraphicsImportComponent ci,
   Rect *bounds
);
```

Parameters

ci
The component instance that identifies your connection to the graphics importer component.

bounds
A pointer to a `Rect` structure describing the bounding rectangle that has been defined for the image.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This is a convenience function. It is implemented by calling `GraphicsImportGetMatrix` (page 1053) and `GraphicsImportGetNaturalBounds` (page 1055) and using the results to calculate the drawing rectangle.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qteffects

qteffects.win

qtgraphics

qtgraphics.win

vrmakepano

Declared In
`ImageCompression.h`

GraphicsImportGetClip

Returns the current clipping region for an imported image.

```
ComponentResult GraphicsImportGetClip (
   GraphicsImportComponent ci,
   RgnHandle *clipRgn
);
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

clipRgn

A handle to the `MacRegion` structure that has been defined as the clipping region for the image. Returns `NIL` if there is no clipping region.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
The caller must dispose of the returned region handle.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GraphicsImportGetColorSyncProfile

Returns a ColorSync profile for an imported image, if one is embedded in the image file.

```
ComponentResult GraphicsImportGetColorSyncProfile (
   GraphicsImportComponent ci,
   Handle *profile
);
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

profile

A pointer to receive a handle containing a ColorSync profile, or `NIL` if the image file does not contain one.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Some graphics importers don't implement this function. The caller is responsible for disposing of the returned handle.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Graphic Import-Export

ImproveYourImage

QTtoCG

Declared In

`ImageCompression.h`

GraphicsImportGetDataFile

Returns the file containing the graphics data for an imported image.

```
ComponentResult GraphicsImportGetDataFile (
   GraphicsImportComponent ci,
   FSSpec *theFile
);
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

theFile

A pointer in which to return the `FSSpec` structure of the file containing the graphics data.

Return Value

See `Error Codes`. Returns `noErr` if there is no error. If the data source is not a file, the function returns `paramErr`.

Discussion

Use this function to get the file system specification record for the file where the imported graphics data resides.

Special Considerations

Graphics importer components use QuickTime data handler components to obtain their data. Applications, however, will use graphics importer functions rather than directly calling a data handler. Besides `GraphicsImportGetDataFile`, these functions include `GraphicsImportSetDataFile` (page 1065), `GraphicsImportSetDataHandle` (page 1066), `GraphicsImportGetDataHandle` (page 1038), `GraphicsImportSetDataReference` (page 1067), `GraphicsImportSetDataReferenceOffsetAndLimit` (page 1068), and `GraphicsImportGetDataReferenceOffsetAndLimit` (page 1041). These functions allow the data source to be a file, a handle, or a QuickTime data reference. You only need to use these functions if you open the graphics importer component directly. You don't need to call them if you use one of the `GetGraphicsImporter`... functions such as `GetGraphicsImporterForDataRef` (page 663). The `GetGraphicsImporter`... functions automatically open the graphics importer component and set its data source.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GraphicsImportGetDataHandle

Returns a handle to imported graphics data.

```
ComponentResult GraphicsImportGetDataHandle (
   GraphicsImportComponent ci,
   Handle *h
);
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

h

A pointer in which to return a handle to the graphics data.

Return Value

See `Error Codes`. Returns `noErr` if there is no error. If the data source is not a handle, the function returns `paramErr`.

Discussion

You use this function to get the handle that the graphics data resides in. The handle belongs to the component instance. You shouldn't dispose of it.

Special Considerations

Graphics importer components use QuickTime data handler components to obtain their data. Applications, however, will use graphics importer functions rather than directly calling a data handler. Besides `GraphicsImportGetDataHandle`, these functions include `GraphicsImportSetDataFile` (page 1065), `GraphicsImportSetDataHandle` (page 1066), `GraphicsImportGetDataFile` (page 1037), `GraphicsImportSetDataReference` (page 1067), `GraphicsImportSetDataReferenceOffsetAndLimit` (page 1068), and `GraphicsImportGetDataReferenceOffsetAndLimit` (page 1041). These functions allow the data source to be a file, a handle, or a QuickTime data reference. You only need to use these functions if you open the graphics importer component directly. You don't need to call them if you use one of the `GetGraphicsImporter`... functions such as `GetGraphicsImporterForDataRef` (page 663). The `GetGraphicsImporter`... functions automatically open the graphics importer component and set its data source.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GraphicsImportGetDataOffsetAndSize

Returns the offset and size of the compressed image data within an imported image file.

```
ComponentResult GraphicsImportGetDataOffsetAndSize (
   GraphicsImportComponent ci,
   unsigned long *offset,
   unsigned long *size
);
```

Parameters

`ci`
: The component instance that identifies your connection to the graphics importer component.

`offset`
: A pointer to a value describing the byte offset of the image data from the beginning of the data source.

`size`
: A pointer to a value describing the size of the image data in bytes.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function returns the offset and size of the actual image data within the data source. By default, the offset returned is 0 and the size returned is the size of the file. However, some graphics import components will override this function to skip over unneeded information at the beginning or end of the file.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
ElectricImageComponent

ElectricImageComponent.win

Declared In
`ImageCompression.h`

GraphicsImportGetDataOffsetAndSize64

Provides a 64-bit version of GraphicsImportGetDataOffsetAndSize.

```
ComponentResult GraphicsImportGetDataOffsetAndSize64 (
   GraphicsImportComponent ci,
   wide *offset,
   wide *size
);
```

Parameters

ci
: The component instance that identifies your connection to the graphics importer component.

offset
: A pointer to a value describing the byte offset of the image data.

size
: A pointer to the size of the data, in bytes.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Format-specific importers may delegate this function, in which case the base importer's implementation will call the 32-bit equivalent, `GraphicsImportGetDataOffsetAndSize` (page 1039). If neither function is implemented by the format-specific importer, then both functions will return an offset of 0 and the full size of the data reference, taking into account any data reference offset and limit.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GraphicsImportGetDataReference

Returns a data reference to imported graphics data.

```
ComponentResult GraphicsImportGetDataReference (
   GraphicsImportComponent ci,
   Handle *dataRef,
   OSType *dataReType
);
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

dataRef

A pointer in which to return a QuickTime data reference. If you don't want this information, pass `NIL`.

dataReType

A pointer to receive the type of the data reference; see `Data References`. If you don't want this information, pass `NIL`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

You use this function to get the data reference that the graphics data resides in. The `GraphicsImportGetDataHandle` (page 1038) and `GraphicsImportGetDataFile` (page 1037) functions call `GraphicsImportGetDataReference` and then manipulate the result accordingly. The caller should dispose of the returned `dataRef`.

Special Considerations

Graphics importer components use QuickTime data handler components to obtain their data. Applications, however, will use graphics importer functions rather than directly calling a data handler. Besides `GraphicsImportGetDataReference`, these functions include `GraphicsImportSetDataFile` (page 1065), `GraphicsImportSetDataHandle` (page 1066), `GraphicsImportGetDataFile` (page 1037), `GraphicsImportSetDataReference` (page 1067), `GraphicsImportSetDataReferenceOffsetAndLimit` (page 1068), and `GraphicsImportGetDataReferenceOffsetAndLimit` (page 1041). These functions allow the data source to be a file, a handle, or a QuickTime data reference. You only need to use these functions if you open the graphics importer component directly. You don't need to call them if you use one of the `GetGraphicsImporter`... functions such as `GetGraphicsImporterForDataRef` (page 663). The `GetGraphicsImporter`... functions automatically open the graphics importer component and set its data source.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GraphicsImportGetDataReferenceOffsetAndLimit

Returns the data reference starting offset and data size limit for an imported image.

```
ComponentResult GraphicsImportGetDataReferenceOffsetAndLimit (
   GraphicsImportComponent ci,
   unsigned long *offset,
   unsigned long *limit
);
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

offset

A pointer to a value specifying the byte offset of the image data from the beginning of the data reference.

limit

The offset of the byte following the last byte of the image data.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function returns the values set by the `GraphicsImportSetDataReferenceOffsetAndLimit` (page 1068) function. By default, the offset is 0 and the limit is `MaxInt` (`2^32 - 1`).

Special Considerations

Graphics importer components use QuickTime data handler components to obtain their data. Applications, however, will use graphics importer functions rather than directly calling a data handler. Besides `GraphicsImportGetDataReferenceOffsetAndLimit`, these functions include `GraphicsImportSetDataFile` (page 1065), `GraphicsImportSetDataHandle` (page 1066), `GraphicsImportGetDataFile` (page 1037), `GraphicsImportSetDataReference` (page 1067), `GraphicsImportSetDataReferenceOffsetAndLimit` (page 1068), and `GraphicsImportGetDataReference` (page 1040). These functions allow the data source to be a file, a handle, or a QuickTime data reference. You only need to use these functions if you open the graphics importer component directly. You don't need to call them if you use one of the `GetGraphicsImporter`... functions such as `GetGraphicsImporterForDataRef` (page 663). The `GetGraphicsImporter`... functions automatically open the graphics importer component and set its data source.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GraphicsImportGetDataReferenceOffsetAndLimit64

Provides a 64-bit version of GraphicsImportGetDataReferenceOffsetAndLimit.

```
ComponentResult GraphicsImportGetDataReferenceOffsetAndLimit64 (
   GraphicsImportComponent ci,
   wide *offset,
   wide *limit
);
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

offset

A pointer to receive a value specifying the offset of the byte data following the last byte of the image data.

limit

A pointer to the data limit.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The only difference between this function and `GraphicsImportGetDataReferenceOffsetAndLimit` (page 1041) is that the `offset` parameter and the `limit` parameter are 64-bit integers instead of 32-bit integers.

Special Considerations

New applications should use this function instead of the 32-bit version.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GraphicsImportGetDefaultClip

Returns the default clipping region for an imported image, if one is stored there.

```
ComponentResult GraphicsImportGetDefaultClip (
   GraphicsImportComponent ci,
   RgnHandle *defaultRgn
);
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

defaultRgn

A pointer to a handle to a `MacRegion` structure to receive the default clipping region.

Return Value

See `Error Codes`. Returns `noErr` if there is no error. Returns `badComponentSelector` if there is no clipping region.

Special Considerations

Most graphics importers don't implement this function. The caller is responsible for disposing of the returned region.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Graphic Import-Export

ImproveYourImage

Declared In
`ImageCompression.h`

GraphicsImportGetDefaultGraphicsMode

Returns the default graphics mode for an imported image, if one is stored there.

```
ComponentResult GraphicsImportGetDefaultGraphicsMode (
   GraphicsImportComponent ci,
   long *defaultGraphicsMode,
   RGBColor *defaultOpColor
);
```

Parameters

`ci`

The component instance that identifies your connection to the graphics importer component.

`defaultGraphicsMode`

A pointer to receive the graphics mode; see `Graphics Transfer Modes`.

`defaultOpColor`

A pointer to receive a color; see `Color Constants`.

Return Value
See `Error Codes`. Returns `noErr` if there is no error. If this function returns `badComponentSelector`, you should assume a mode of `ditherCopy`.

Special Considerations

Most graphics importers don't implement this function.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Graphic Import-Export

ImproveYourImage

Declared In
`ImageCompression.h`

GraphicsImportGetDefaultMatrix

Returns the default matrix for an imported image, if one is stored there.

```
ComponentResult GraphicsImportGetDefaultMatrix (
   GraphicsImportComponent ci,
   MatrixRecord *defaultMatrix
);
```

Parameters

`ci`

The component instance that identifies your connection to the graphics importer component.

`defaultMatrix`

Receives a matrix record.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
If this function returns `badComponentSelector`, you should assume an identity matrix.

Special Considerations
Most graphics importers don't implement this function.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Graphic Import-Export

ImproveYourImage

Declared In
`ImageCompression.h`

GraphicsImportGetDefaultSourceRect

Returns the default source rectangle for an imported image, if one is stored there.

```
ComponentResult GraphicsImportGetDefaultSourceRect (
   GraphicsImportComponent ci,
   Rect *defaultSourceRect
);
```

Parameters

`ci`

The component instance that identifies your connection to the graphics importer component.

defaultSourceRect

Pointer to receive a `Rect` structure that describes the default source rectangle.

Return Value

See `Error Codes`. Returns `noErr` if there is no error. If this function returns `badComponentSelector`, the source rectangle is equal to the image's natural bounds.

Special Considerations

Most graphics importers don't implement this function.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Graphic Import-Export

ImproveYourImage

Declared In

`ImageCompression.h`

GraphicsImportGetDestinationColorSyncProfileRef

Retrieves a ColorSync profile from a graphics importer component.

```
ComponentResult GraphicsImportGetDestinationColorSyncProfileRef (
   GraphicsImportComponent ci,
   CMProfileRef *destinationProfileRef
);
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

destinationProfileRef

On return, a pointer to an opaque struct containing a ColorSync profile.

Return Value

See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 6.4.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`ImageCompression.h`

GraphicsImportGetDestRect

Returns the destination rectangle for an imported image.

```
ComponentResult GraphicsImportGetDestRect (
   GraphicsImportComponent ci,
   Rect *destRect
);
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

destRect

A pointer to receive a `Rect` structure that describes the destination rectangle.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

If the source rectangle is equal to the natural bounds, this function is equivalent to `GraphicsImportGetBoundsRect` (page 1035).

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GraphicsImportGetExportImageTypeList

Returns information about available export formats for a graphics importer.

```
ComponentResult GraphicsImportGetExportImageTypeList (
   GraphicsImportComponent ci,
   void *qtAtomContainerPtr
);
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

qtAtomContainerPtr

A pointer to a QuickTime atom container that is to receive the type list.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function creates and returns a QuickTime atom container of type `'expo'` containing information about the file types that can be exported by the graphics import component. For each file type, the atom container contains the following child atoms: `'ftyp'`, the exported file type; `'mime'[atom]`, the MIME type for this format (optional); `'ext '`, the suggested file extension for this format; and `'desc'`, a human-readable name for this format. The `'ftyp'` atom contains an `OSType`; the other atoms contain nonterminated strings.

Special Considerations

It is the responsibility of the caller to dispose of the `'expo'` atom container.

Version Notes

In QuickTime 3, the supported export file types are `kQTFileTypePicture`, `kQTFileTypeQuickTimeImage`, `kQTFileTypeBMP`, `kQTFileTypeJPEG`, and `kQTFileTypePhotoShop`. In QuickTime 4, the generic graphics importer builds this atom container from the values returned by the installed graphics exporter components.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qtgraphimp

qtgraphimp.win

Declared In

`ImageCompression.h`

GraphicsImportGetExportSettingsAsAtomContainer

Retrieves settings for image files exported by the graphics importer.

```
ComponentResult GraphicsImportGetExportSettingsAsAtomContainer (
   GraphicsImportComponent ci,
   void *qtAtomContainerPtr
);
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

qtAtomContainerPtr

A pointer to a QuickTime atom container that is to receive the settings information.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function creates and returns a new QuickTime atom container which holds information about how images will be saved by `GraphicsImportExportImageFile` (page 1031).

Special Considerations

It is the responsibility of the caller to dispose of this atom container.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Fiendishthngs

Declared In

`ImageCompression.h`

GraphicsImportGetFlags

Returns the current flags of a graphics importer component.

```
ComponentResult GraphicsImportGetFlags (
   GraphicsImportComponent ci,
   long *flags
);
```

Parameters

ci

The component instance that identifies your connection to a graphics importer component.

flags

Pointer to a long integer to receive the current flags (see below). See these constants:
`kGraphicsImporterDontDoGammaCorrection`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

ColorMatching

Declared In

`ImageCompression.h`

GraphicsImportGetGenericColorSyncProfile

Retrieves the generic colorsync profile for a graphics importer component.

```
ComponentResult GraphicsImportGetGenericColorSyncProfile (
   GraphicsImportComponent ci,
   OSType pixelFormat,
   void *reservedSetToNULL,
   UInt32 flags,
   Handle *genericColorSyncProfileOut
);
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

pixelFormat

See `Pixel Formats` in the QuickTime API Reference.

reservedSetToNULL

Pass `NIL`.

flags

Currently not used.

genericColorSyncProfileOut

A handle to the the generic `colorsync` profile for the graphics importer.

Return Value

See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 6.4.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`ImageCompression.h`

GraphicsImportGetGraphicsMode

Returns the graphics transfer mode for an imported image.

```
ComponentResult GraphicsImportGetGraphicsMode (
   GraphicsImportComponent ci,
   long *graphicsMode,
   RGBColor *opColor
);
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

graphicsMode

A pointer to a long integer; see `Graphics Transfer Modes`. The function returns the QuickDraw graphics transfer mode setting for the image. Set to `NIL` if you are not interested in this information.

opColor

A pointer to an `RGBColor` structure. The function returns the color currently specified for blend and transparent operations. Set to `NIL` if you are not interested in this information.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Use this function to find out the current graphics transfer mode and color to use for blending and transparent operations. The default graphics mode is `ditherCopy` and the default opColor is 50% gray.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GraphicsImportGetGWorld

Returns the current graphics port and device for drawing an imported image.

```
ComponentResult GraphicsImportGetGWorld (
   GraphicsImportComponent ci,
   CGrafPtr *port,
   GDHandle *gd
);
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

port

Returns a pointer to the `CGrafPort` structure for the current destination graphics port. Set to `NIL` if you are not interested in this information.

gd

Returns a pointer to the `GDevice` structure for the destination graphics device. Set to `NIL` if you are not interested in this information.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function returns the graphics port and device that will be used to draw the image. The graphics world is initialized to the current port and device when the graphics importer component is opened.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GraphicsImportGetImageCount

Returns the number of images in an imported image file.

```
ComponentResult GraphicsImportGetImageCount (
   GraphicsImportComponent ci,
   unsigned long *imageCount
);
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

imageCount

Points to a variable to receive the number of images.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Most image file formats don't support multiple images. Of the image formats supported by QuickTime 4, however, TIFF files can support multiple images, Photoshop files can contain multiple layers and FlashPix files can contain multiple resolutions. The base graphics importer returns a count of 1.

Special Considerations

Format-specific importers for multiple-image formats should override this function; other importers should delegate it.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Graphic Import-Export

qtgraphics

qtgraphics.win

qtmultiimage

qtmultiimage.win

Declared In

`ImageCompression.h`

GraphicsImportGetImageDescription

Returns image description information for an imported image.

```
ComponentResult GraphicsImportGetImageDescription (
   GraphicsImportComponent ci,
   ImageDescriptionHandle *desc
);
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

desc

Points to a handle to an `ImageDescription` structure.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function returns an `ImageDescription` structure containing information such as the format of the compressed data, its bit depth, natural bounds, and resolution.

Special Considerations

The caller is responsible for disposing of the returned image description handle.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CTMDemo

Graphic Import-Export

ImagesToQTMovie

ImproveYourImage

TextureRange

Declared In

`ImageCompression.h`

GraphicsImportGetImageIndex

Returns the current image index for an imported image.

```
ComponentResult GraphicsImportGetImageIndex (
   GraphicsImportComponent ci,
   unsigned long *imageIndex
);
```

Parameters

`ci`

The component instance that identifies your connection to the graphics importer component.

`imageIndex`

Points to a variable to receive the image index. Image indexes are one-based; 0 is considered a special index by some importers, and treated the same as 1 by others. The default image index is 1.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The base graphics importer implements this function. Format-specific importers should delegate it.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GraphicsImportGetMatrix

Returns the transformation matrix to be used for drawing an imported image.

```
ComponentResult GraphicsImportGetMatrix (
   GraphicsImportComponent ci,
   MatrixRecord *matrix
);
```

Parameters

`ci`

The component instance that identifies your connection to the graphics importer component.

matrix

A pointer to a `MatrixRecord` structure that defines the transformation matrix that applies to the image.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The transformation matrix is initialized to the identity matrix when the graphics import component is instantiated.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Graphic Import-Export

ImproveYourImage

qtgraphics

qtgraphics.win

Declared In

`ImageCompression.h`

GraphicsImportGetMetaData

Extracts user data from an imported image file.

```
ComponentResult GraphicsImportGetMetaData (
   GraphicsImportComponent ci,
   void *userData
);
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

userData

A pointer to a `UserDataRecord` structure.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

You may create a new user data structure by calling `NewUserData` (page 1415). Alternatively, you can obtain a pointer to an existing one by calling `GetMovieUserData` (page 225), `GetTrackUserData` (page 1617) or `GetMediaUserData` (page 1595). If the user data passed to `GraphicsImportGetMetaData` belongs to a movie, track or media, then whatever user data is extracted will be added to that movie, track or media.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code
Graphic Import-Export

ImproveYourImage

Declared In
`ImageCompression.h`

GraphicsImportGetMIMETypeList

Returns a list of MIME types supported by the graphics importer component.

```
ComponentResult GraphicsImportGetMIMETypeList (
   GraphicsImportComponent ci,
   void *qtAtomContainerPtr
);
```

Parameters

`ci`

Specifies an instance of a graphics importer component.

`qtAtomContainerPtr`

A pointer to an atom container that holds a series of atom triplets for each MIME type, including an atom of type `'mime'[atom]` that contains a list of MIME types supported by the graphics import component.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Your graphics import component can support MIME types that correspond to graphics formats it supports. To make a list of these MIME types available to applications or other software, it must implement `GraphicsImportGetMIMETypeList`. To indicate that your graphics import component supports this function, set the `hasMovieImportMIMEList` flag in the `componentFlags` field of your component's `ComponentDescription` structure.

Special Considerations

This function does not access any file-specific information.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GraphicsImportGetNaturalBounds

Returns the bounding rectangle of an imported image.

```
ComponentResult GraphicsImportGetNaturalBounds (
   GraphicsImportComponent ci,
   Rect *naturalBounds
);
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

naturalBounds

A pointer to a `Rect` structure that describes the size of the bounding rectangle for the image.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Use this function to determine the native size of the image associated with a graphics importer component. The natural bounds are always zero-based. This is a convenience function that simply calls `GraphicsImportGetImageDescription` (page 1052) and extracts the `width` and height fields.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

DelegateOnlyComponent

Graphic Import-Export

ImproveYourImage

qtgraphics

qtgraphics.win

Declared In

`ImageCompression.h`

GraphicsImportGetOverrideSourceColorSyncProfileRef

Retrieves the override ColorSync profile for a graphics importer component.

```
ComponentResult GraphicsImportGetOverrideSourceColorSyncProfileRef (
   GraphicsImportComponent ci,
   CMProfileRef *outOverrideSourceProfileRef
);
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

outOverrideSourceProfileRef

A pointer to an opaque struct containing a ColorSync profile.

Return Value

See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 6.4.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`ImageCompression.h`

GraphicsImportGetProgressProc

Returns the current progress function for a graphics import operation.

```
ComponentResult GraphicsImportGetProgressProc (
   GraphicsImportComponent ci,
   ICMProgressProcRecordPtr progressProc
);
```

Parameters

ci
: The component instance that identifies your connection to the graphics importer component.

progressProc
: A pointer to an `ICMProgressProc` callback.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
By default, graphics import components have no progress functions.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GraphicsImportGetQuality

Returns the image quality value for an imported image.

```
ComponentResult GraphicsImportGetQuality (
   GraphicsImportComponent ci,
   CodecQ *quality
);
```

Parameters

ci
: The component instance that identifies your connection to the graphics importer component.

quality

A pointer to a constant (see below) that defines the currently specified quality value. See these constants:

`codecMinQuality`

`codecLowQuality`

`codecNormalQuality`

`codecHighQuality`

`codecMaxQuality`

`codecLosslessQuality`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The quality value indicates how precisely the decompressor will decompress the image data. Some decompressors may choose to ignore some image data to improve decompression speed.

Version Notes

With QuickTime 3 the default quality is `codecHighQuality`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GraphicsImportGetSourceRect

Returns the current source rectangle for an imported image.

```
ComponentResult GraphicsImportGetSourceRect (
   GraphicsImportComponent ci,
   Rect *sourceRect
);
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

sourceRect

A pointer to a `Rect` structure that defines the source rectangle currently specified for the image.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function returns the current source rectangle, as specified by `GraphicsImportSetSourceRect` (page 1078). The default source rectangle is the image's natural bounds.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code
CarbonQTGraphicImport

Declared In
`ImageCompression.h`

GraphicsImportReadData

Reads imported image data.

```
ComponentResult GraphicsImportReadData (
   GraphicsImportComponent ci,
   void *dataPtr,
   unsigned long dataOffset,
   unsigned long dataSize
);
```

Parameters

ci
: The component instance that identifies your connection to the graphics importer component.

dataPtr
: A pointer to a memory block to receive the data.

dataOffset
: The offset of the image data within the data reference. The function begins reading image data from this offset.

dataSize
: The number of bytes of image data to read.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function communicates with the appropriate data handler to retrieve image data. Typically, only developers of graphics importer components will need to use this function. This function should always be used to retrieve data from the data source, rather than reading the data directly. This function automatically honors any offset and limit values set with `GraphicsImportSetDataReferenceOffsetAndLimit` (page 1068). For instance, if the offset is set to 100 and `GraphicsImportReadData` is called to read bytes from `dataOffset` 5, it will return bytes starting at actual offset 105.

Special Considerations

This function is used by format-specific graphics import components to read data from the data source. It is implemented by the base graphics importer.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
ElectricImageComponent
ElectricImageComponent.win

Declared In

`ImageCompression.h`

GraphicsImportReadData64

Provides a 64-bit version of GraphicsImportReadData.

```
ComponentResult GraphicsImportReadData64 (
   GraphicsImportComponent ci,
   void *dataPtr,
   const wide *dataOffset,
   unsigned long dataSize
);
```

Parameters

`ci`

The component instance that identifies your connection to the graphics importer component.

`dataPtr`

A pointer to a memory block to receive the data.

`dataOffset`

A pointer to the offset of the image data within the data reference.

`dataSize`

The number of bytes of image data to read.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function is a 64-bit analog of `GraphicsImportReadData` (page 1059). Format-specific importers may call either version.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GraphicsImportSaveAsPicture

Creates a QuickDraw picture file for an imported image.

```
ComponentResult GraphicsImportSaveAsPicture (
   GraphicsImportComponent ci,
   const FSSpec *fss,
   ScriptCode scriptTag
);
```

Parameters

`ci`

The component instance that identifies your connection to the graphics importer component.

fss

A pointer to an FSSpec structure that defines the file to receive the image.

scriptTag

The script system in which the file name is to be displayed; see Localization Codes. If you have established the name and location of the file using one of the Standard File Package functions, use the script code returned in the reply record (reply.sfScript). Otherwise, specify the system script by setting the scriptTag parameter to the value smSystemScript. See *Inside Macintosh: Files* for more information about script specifications.

Return Value

See Error Codes. Returns noErr if there is no error.

Discussion

This function creates a new QuickDraw picture file containing the image currently in use by the graphics import component. If possible, the image will remain in the compressed format. For example, if the image is from a JFIF file, the picture will contain compressed JPEG data. Applications that only wish to draw the image can use GraphicsImportDraw (page 1030).

Special Considerations

Graphics import components can save data in several formats, including QuickDraw pictures and QuickTime Image files. This capability is only needed by applications that perform file format translation.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

ImageCompression.h

GraphicsImportSaveAsPictureToDataRef

Creates a storage location that contains a QuickDraw picture for an imported image.

```
ComponentResult GraphicsImportSaveAsPictureToDataRef (
   GraphicsImportComponent ci,
   Handle dataRef,
   OSType dataRefType
);
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

dataRef

A data reference that specifies a storage location to which the picture is to be saved.

dataRefType

The type of the data reference.

Return Value

See Error Codes in the QuickTime API Reference. Returns noErr if there is no error.

Discussion

This function saves the imported image as a QuickDraw picture into a storage location specified through a data reference. You can use Data Reference Utilities to create a data reference for a file that has long or Unicode file name. This function is equivalent to `GraphicsImporterSaveAsPictureFile`.

Version Notes

Introduced in QuickTime 6.4.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`ImageCompression.h`

GraphicsImportSaveAsQuickTimeImageFile

Creates a QuickTime Image file of an imported image.

```
ComponentResult GraphicsImportSaveAsQuickTimeImageFile (
   GraphicsImportComponent ci,
   const FSSpec *fss,
   ScriptCode scriptTag
);
```

Parameters

`ci`

The component instance that identifies your connection to the graphics importer component.

`fss`

A pointer to the `FSSpec` that defines the file to receive the image.

`scriptTag`

The script system in which the file name is to be displayed; see `Localization Codes`. If you have established the name and location of the file using one of the Standard File Package functions, use the script code returned in the reply record (`reply.sfScript`). Otherwise, specify the system script by setting the `scriptTag` parameter to the value `smSystemScript`. See *Inside Macintosh: Files* for more information about script specifications.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function creates a new QuickTime Image file containing the image currently in use by the graphics import component. If possible, the image remains in the compressed format. For example, if the image is from a JFIF file, the QuickTime Image file will contain compressed JPEG data.

Special Considerations

Graphics import components can save data in several formats, including QuickDraw pictures and QuickTime Image files. This capability is only needed by applications that perform file format translation. Applications that only wish to draw the image can use the `GraphicsImportDraw` (page 1030) function.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GraphicsImportSaveAsQuickTimeImageFileToDataRef

Creates a storage location that contains a QuickTime image of an imported image.

```
ComponentResult GraphicsImportSaveAsQuickTimeImageFileToDataRef (
   GraphicsImportComponent ci,
   Handle dataRef,
   OSType dataRefType
);
```

Parameters

ci
The component instance that identifies your connection to the graphics importer component.

dataRef
A data reference that specifies a storage location to which the picture is to be saved.

dataRefType
The type of the data reference.

Return Value
See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error.

Discussion
This function saves the imported image as a QuickTime image into a storage location specified through a data reference. You can use data reference functions to create a data reference for a file that has long or Unicode file name. This function is equivalent to `GraphicsImportSaveAsQuickTimeImageFile`.

Version Notes
Introduced in QuickTime 6.4.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`ImageCompression.h`

GraphicsImportSetBoundsRect

Defines the rectangle in which to draw an imported image.

```
ComponentResult GraphicsImportSetBoundsRect (
   GraphicsImportComponent ci,
   const Rect *bounds
);
```

Parameters

ci
The component instance that identifies your connection to the graphics importer component.

bounds

A pointer to a `Rect` structure that describes the bounding rectangle into which the image will be drawn.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

You use this function to define the rectangle into which the graphics image should be drawn. The function creates a transformation matrix to map the image's natural bounds to the specified bounds and then calls `GraphicsImportSetMatrix` (page 1075).

Special Considerations

Because this function affects the transformation matrix, you should use `GraphicsImportSetMatrix` (page 1075) instead of this function if you also need to specify more complex transformations of the matrix.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CarbonQTGraphicImport

qtstreamsplicer.win

ThreadsImporter

vrmakepano

VRMakePano Library

Declared In

`ImageCompression.h`

GraphicsImportSetClip

Defines the clipping region for drawing an imported image.

```
ComponentResult GraphicsImportSetClip (
   GraphicsImportComponent ci,
   RgnHandle clipRgn
);
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

clipRgn

A handle to a `MacRegion` structure that defines the clipping region in the destination coordinate system. Set to `NIL` to disable clipping. The graphics import component makes a copy of this region.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Because all drawing operations ignore the port clipping region, you must use this function to clip an image. The graphics importer component draws only that portion of the image that lies within the specified clipping region.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

DropDraw

QTGraphicsImport

Declared In

`ImageCompression.h`

GraphicsImportSetDataFile

Specifies the file that contains imported graphics data.

```
ComponentResult GraphicsImportSetDataFile (
   GraphicsImportComponent ci,
   const FSSpec *theFile
);
```

Parameters

`ci`
: The component instance that identifies your connection to the graphics importer component.

`theFile`
: A pointer to an `FSSpec` structure that defines the file containing the graphics data. The returned file will be opened for read access.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Special Considerations

Graphics importer components use QuickTime data handler components to obtain their data. Applications, however, will use graphics importer functions rather than directly calling a data handler. Besides `GraphicsImportSetDataFile`, these functions include `GraphicsImportGetDataFile` (page 1037), `GraphicsImportSetDataHandle` (page 1066), `GraphicsImportGetDataHandle` (page 1038), `GraphicsImportSetDataReference` (page 1067), `GraphicsImportSetDataReferenceOffsetAndLimit` (page 1068), and `GraphicsImportGetDataReferenceOffsetAndLimit` (page 1041). These functions allow the data source to be a file, a handle, or a QuickTime data reference. You only need to use these functions if you open the graphics importer component directly. You don't need to call them if you use one of the `GetGraphicsImporter`... functions such as `GetGraphicsImporterForDataRef` (page 663). The `GetGraphicsImporter`... functions automatically open the graphics importer component and set its data source.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GraphicsImportSetDataHandle

Specifies the handle that references imported graphics data.

```
ComponentResult GraphicsImportSetDataHandle (
   GraphicsImportComponent ci,
   Handle h
);
```

Parameters

`ci`
The component instance that identifies your connection to the graphics importer component.

`h`
Specifies a handle containing graphics data. The format of the data in the handle is the same as that found in a file.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
The graphics importer component doesn't make a copy of this data. Therefore, you must not dispose this handle until the graphics importer has been closed.

Special Considerations
Graphics importer components use QuickTime data handler components to obtain their data. Applications, however, will use graphics importer functions rather than directly calling a data handler. Besides `GraphicsImportSetDataHandle`, these functions include `GraphicsImportGetDataFile` (page 1037), `GraphicsImportSetDataFile` (page 1065), `GraphicsImportGetDataHandle` (page 1038), `GraphicsImportSetDataReference` (page 1067), `GraphicsImportSetDataReferenceOffsetAndLimit` (page 1068), and `GraphicsImportGetDataReferenceOffsetAndLimit` (page 1041). These functions allow the data source to be a file, a handle, or a QuickTime data reference. You only need to use these functions if you open the graphics importer component directly. You don't need to call them if you use one of the `GetGraphicsImporter`... functions such as `GetGraphicsImporterForDataRef` (page 663). The `GetGraphicsImporter`... functions automatically open the graphics importer component and set its data source.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CarbonQTGraphicImport
qtdataref
qtgraphimp.win

qtmakemovie
ThreadsImportMovie

Declared In
`ImageCompression.h`

GraphicsImportSetDataReference

Specifies the data reference for imported graphics data.

```
ComponentResult GraphicsImportSetDataReference (
   GraphicsImportComponent ci,
   Handle dataRef,
   OSType dataReType
);
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

dataRef

A handle to a QuickTime data reference.

dataReType

The data reference type. See `Data References`.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Applications typically don't use this function. The `GraphicsImportSetDataFile` (page 1065) and `GraphicsImportSetDataHandle` (page 1066) functions both call this function, with the appropriate data reference and data reference type. This function makes a copy of the passed data reference, so it is safe to dispose of the handle immediately after the call.

Special Considerations

Graphics importer components use QuickTime data handler components to obtain their data. Applications, however, will use graphics importer functions rather than directly calling a data handler. Besides `GraphicsImportSetDataReference`, these functions include `GraphicsImportGetDataFile` (page 1037), `GraphicsImportSetDataHandle` (page 1066), `GraphicsImportGetDataHandle` (page 1038), `GraphicsImportSetDataFile` (page 1065), `GraphicsImportSetDataReferenceOffsetAndLimit` (page 1068), and `GraphicsImportGetDataReferenceOffsetAndLimit` (page 1041). These functions allow the data source to be a file, a handle, or a QuickTime data reference. You only need to use these functions if you open the graphics importer component directly. You don't need to call them if you use one of the `GetGraphicsImporter`... functions such as `GetGraphicsImporterForDataRef` (page 663). The `GetGraphicsImporter`... functions automatically open the graphics importer component and set its data source.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GraphicsImportSetDataReferenceOffsetAndLimit

Specifies the data reference starting offset and data size limit for an imported image.

```
ComponentResult GraphicsImportSetDataReferenceOffsetAndLimit (
   GraphicsImportComponent ci,
   unsigned long offset,
   unsigned long limit
);
```

Parameters

`ci`
: The component instance that identifies your connection to the graphics importer component.

`offset`
: The byte offset of the image data from the beginning of the data reference.

`limit`
: A pointer to a value specifying the offset of the byte following the last byte of the image data.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

A data reference typically refers to an entire file. However, there are times when the data being referenced is embedded in a larger file. In these cases, it is necessary to indicate where the data begins in the data reference and where it ends. This function lets you specify the starting offset and ending offset. All requests to read data are then relative to the specified offset, and are pinned to the data size, so the graphics import component cannot accidentally read outside the end (or beginning) of the segment.

Special Considerations

Graphics importer components use QuickTime data handler components to obtain their data. Applications, however, will use graphics importer functions rather than directly calling a data handler. Besides `GraphicsImportSetDataReferenceOffsetAndLimit`, these functions include `GraphicsImportGetDataFile` (page 1037), `GraphicsImportSetDataHandle` (page 1066), `GraphicsImportGetDataHandle` (page 1038), `GraphicsImportSetDataReference` (page 1067), `GraphicsImportSetDataFile` (page 1065), and `GraphicsImportGetDataReferenceOffsetAndLimit` (page 1041). These functions allow the data source to be a file, a handle, or a QuickTime data reference. You only need to use these functions if you open the graphics importer component directly. You don't need to call them if you use one of the `GetGraphicsImporter`... functions such as `GetGraphicsImporterForDataRef` (page 663). The `GetGraphicsImporter`... functions automatically open the graphics importer component and set its data source.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GraphicsImportSetDataReferenceOffsetAndLimit64

Provides a 64-bit version of GraphicsImportSetDataReferenceOffsetAndLimit.

```
ComponentResult GraphicsImportSetDataReferenceOffsetAndLimit64 (
   GraphicsImportComponent ci,
   const wide *offset,
   const wide *limit
);
```

Parameters

ci

A component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

offset

A pointer to a value specifying the byte offset of the image data from the beginning of the data source.

limit

A pointer to a value specifying the offset of the byte following the last byte of the image data.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function is a 64-bit analog of `GraphicsImportSetDataReferenceOffsetAndLimit` (page 1068).

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GraphicsImportSetDestinationColorSyncProfileRef

Sets the ColorSync profile for a graphics importer component.

```
ComponentResult GraphicsImportSetDestinationColorSyncProfileRef (
   GraphicsImportComponent ci,
   CMProfileRef newDestinationProfileRef
);
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

newDestinationProfileRef

A pointer to an opaque struct containing a ColorSync profile.

Return Value

See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 6.4.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`ImageCompression.h`

GraphicsImportSetDestRect

Sets the destination rectangle for a graphics import operation.

```
ComponentResult GraphicsImportSetDestRect (
   GraphicsImportComponent ci,
   const Rect *destRect
);
```

Parameters

`ci`
The component instance that identifies your connection to the graphics importer component.

`destRect`
Points to a `Rect` structure.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Use this function to define the rectangle into which the extracted source rectangle should be drawn. This function creates a transformation matrix to map the source rectangle to the specified destination rectangle and then calls the `GraphicsImportSetMatrix` (page 1075) function.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GraphicsImportSetExportSettingsFromAtomContainer

Determines settings for the export of imported image files.

```
ComponentResult GraphicsImportSetExportSettingsFromAtomContainer (
   GraphicsImportComponent ci,
   void *qtAtomContainer
);
```

Parameters

`ci`
The component instance that identifies your connection to the graphics importer component.

`qtAtomContainer`
A pointer to a QuickTime atom container that holds new settings information.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function extracts export settings from a QuickTime atom container. These settings configure how images will be saved by `GraphicsImportExportImageFile` (page 1031).

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GraphicsImportSetFlags

Sets the flags for a graphics importer component.

```
ComponentResult GraphicsImportSetFlags (
   GraphicsImportComponent ci,
   long flags
);
```

Parameters

`ci`
The component instance that identifies your connection to the graphics importer component.

`flags`
The new flags (see below) to use. See these constants:
`kGraphicsImporterDontDoGammaCorrection`

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
ColorMatching
DropDraw
ImproveYourImage
SampleDS

Declared In
`ImageCompression.h`

GraphicsImportSetGraphicsMode

Sets the graphics transfer mode for an imported image.

```
ComponentResult GraphicsImportSetGraphicsMode (
   GraphicsImportComponent ci,
   long graphicsMode,
   const RGBColor *opColor
);
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

graphicsMode

The graphics transfer mode to use for drawing the image; see `Graphics Transfer Modes`.

opColor

A pointer to an `RGBColor` structure that describes the color to use for blending and transparent operations.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Use this function to specify the graphics transfer mode and color to use for blending and transparent operations.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Graphic Import-Export

ImproveYourImage

Declared In

`ImageCompression.h`

GraphicsImportSetGWorld

Sets the graphics port and device for drawing an imported image.

```
ComponentResult GraphicsImportSetGWorld (
   GraphicsImportComponent ci,
   CGrafPtr port,
   GDHandle gd
);
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

port

A pointer to the `CGrafPort` structure that defines the destination graphics port or graphics world. Set to `NIL` to use the current port.

gd

A handled to the `GDevice` structure that defines the destination graphics device. Set to `NIL` to use the current device. If the `port` parameter specifies a graphics world, set this parameter to `NIL` to use that graphics world's device.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The graphics world is initialized to the current port and device when the graphics importer component is opened. Use this function to select another port or device.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Graphic Import-Export

ImproveYourImage

qtgraphics.win

qtstreamsplicer.win

vrmakepano

Declared In

`ImageCompression.h`

GraphicsImportSetImageIndex

Specifies the image index for an imported image.

```
ComponentResult GraphicsImportSetImageIndex (
   GraphicsImportComponent ci,
   unsigned long imageIndex
);
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

imageIndex

The image index. Image indexes are one-based; 0 is considered a special index by some importers, and treated the same as 1 by others. The default image index is 1.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The base graphics importer ensures that the image index is no greater than the image count returned by `GraphicsImportGetImageCount` (page 1051).

Special Considerations

The base graphics importer implements this function. Format-specific importers should delegate it.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Graphic Import-Export

ImproveYourImage

qtgraphics.win

qtmultiimage

qtmultiimage.win

Declared In

`ImageCompression.h`

GraphicsImportSetImageIndexToThumbnail

Looks for a graphics subimage that contains a thumbnail.

```
ComponentResult GraphicsImportSetImageIndexToThumbnail (
   GraphicsImportComponent ci
);
```

Parameters

`ci`

The component instance that identifies your connection to the graphics importer component.

Return Value

See `Error Codes`. If the function does not find a thumbnail, it returns `noThumbnailFoundErr`. It returns `noErr` if there is no error.

Discussion

This function looks for a subimage that contains a thumbnail. If the function finds one, it sets the image index to that subimage. The base graphics importer's implementation of `SetImageIndexToThumbnail` works by looking for the first image index that returns a `kQTIndexedImageType` metadata item containing the `kQTIndexedImageIsThumbnail` tag. Format-specific graphics importers may override this process with more efficient algorithms.

Version Notes

Introduced in QuickTime 6.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`ImageCompression.h`

GraphicsImportSetMatrix

Defines the transformation matrix to use for drawing an imported image.

```
ComponentResult GraphicsImportSetMatrix (
   GraphicsImportComponent ci,
   const MatrixRecord *matrix
);
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

matrix

A pointer to a matrix structure that specifies how to transform the image during decompression. For example, you can use a transformation matrix to scale or rotate the image. To set the matrix to identity, pass `NIL` in this parameter.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function establishes the transformation matrix to be applied to an image, which determines where and how it will be drawn.

Special Considerations

This function affects the bounding rectangle defined for the image. You can specify where an image will be drawn by setting either a transformation matrix or a bounding rectangle, but it is usually more convenient for applications to set a bounding rectangle using the `GraphicsImportSetBoundsRect` (page 1063) function.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

DropDraw

Graphic Import-Export

ImproveYourImage

qtgraphics

qtgraphics.win

Declared In

`ImageCompression.h`

GraphicsImportSetOverrideSourceColorSyncProfileRef

Sets the override ColorSync profile for a graphics importer component.

```
ComponentResult GraphicsImportSetOverrideSourceColorSyncProfileRef (
   GraphicsImportComponent ci,
   CMProfileRef newOverrideSourceProfileRef
);
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

newOverrideSourceProfileRef

A pointer to an opaque struct containing a ColorSync profile.

Return Value

See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 6.4.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`ImageCompression.h`

GraphicsImportSetProgressProc

Installs a progress procedure to call while drawing an imported image.

```
ComponentResult GraphicsImportSetProgressProc (
   GraphicsImportComponent ci,
   ICMProgressProcRecordPtr progressProc
);
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

progressProc

Points to an `ICMProgressProc` callback. If you pass a value of -1, QuickTime provides a standard progress function. If you want to remove the existing progress function, pass `NIL`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function sets a progress function that will be installed in the image decompression sequence used to draw the image.

Special Considerations

If your progress function does any drawing, you should take care to set a safe graphics state before doing so, and to restore the graphics state afterwards. In particular, the current graphics device may be an offscreen device.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qtdataexchange
qtdataexchange.win
ThreadsExporter
ThreadsImporter

Declared In
`ImageCompression.h`

GraphicsImportSetQuality

Sets the image quality value for an imported image.

```
ComponentResult GraphicsImportSetQuality (
   GraphicsImportComponent ci,
   CodecQ quality
);
```

Parameters

`ci`
The component instance that identifies your connection to the graphics importer component.

`quality`
Contains a constant (see below) that defines the desired image quality for decompression. Values for this parameter are on the same scale as compression quality. See these constants:
- `codecMinQuality`
- `codecLowQuality`
- `codecNormalQuality`
- `codecHighQuality`
- `codecMaxQuality`
- `codecLosslessQuality`

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
The quality parameter controls how precisely the decompressor decompresses the image data. Some decompressors may choose to ignore some image data to improve decompression speed.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CTMClip
CTMDemo
TexturePerformanceDemo

TextureRange

ThreadsImporter

Declared In
`ImageCompression.h`

GraphicsImportSetSourceRect

Sets the source rectangle to use for an imported image.

```
ComponentResult GraphicsImportSetSourceRect (
   GraphicsImportComponent ci,
   const Rect *sourceRect
);
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

sourceRect

A pointer to a `Rect` structure defining the portion of the image to decompress. This rectangle must lie within the boundary rectangle of the source image. Set to `NIL` to use the entire image.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function provides a way to use only a portion of the source image.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CarbonQTGraphicImport

DropDraw

Declared In
`ImageCompression.h`

GraphicsImportValidate

Validates image data for a data reference to an imported image.

```
ComponentResult GraphicsImportValidate (
   GraphicsImportComponent ci,
   Boolean *valid
);
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

valid

Pointer to a Boolean value. On return, this parameter is set to TRUE if the the graphics importer component can draw the data reference. If the graphics importer component cannot draw the data reference, this parameter is set to FALSE.

Return Value

See `Error Codes`. Returns `noErr` if there is no error. Not all graphics importer components implement this function. A component that does not implement the function will return the `badComponentSelector` result code. This does not indicate that the file is valid or invalid.

Discussion

This function allows a graphics importer component to determine if its current data reference contains valid image data. For example, a JFIF graphics importer component might check for the presence of a JFIF marker at the start of the data stream. This function is provided for applications to use to determine what type of image data a particular file may contain. Sometimes a file may not have the correct file type or file extension. In this case, the application will not know which graphics importer component to use. By iterating through all graphics importer components and calling `GraphicsImportValidate` for each one, it may be possible to locate a graphics importer component that can draw the specified file.

Special Considerations

`GraphicsImportValidate` does not perform an exhaustive check on the file. It is possible for `GraphicsImportValidate` to claim a data reference is valid but for `GraphicsImportDraw` (page 1030) to return an error due to bad data. Format-specific importers that implement the `GraphicsImportValidate` call should have the `canMovieImportValidateFile` bit set in the `flags` field of their `ComponentDescription` structures.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ImageCompression.h`

GraphicsImportWillUseColorMatching

Asks whether GraphicsImportDraw will use color matching if called with the current importer settings.

```
ComponentResult GraphicsImportWillUseColorMatching (
   GraphicsImportComponent ci,
   Boolean *outWillMatch
);
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

outWillMatch

On return, a pointer to a Boolean set to TRUE if the graphics importer will use color matching, FALSE otherwise.

Return Value

See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 6.4.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`ImageCompression.h`

Callbacks

ModalFilterYDProc

Determines how the Dialog Manager filters events.

```
typedef Boolean (*ModalFilterYDProcPtr) (DialogPtr theDialog, EventRecord *theEvent,
 short *itemHit, void *yourDataPtr);
```

If you name your function `MyModalFilterYDProc`, you would declare it this way:

```
Boolean MyModalFilterYDProc (
    DialogPtr      theDialog,
    EventRecord    *theEvent,
    short          *itemHit,
    void           *yourDataPtr );
```

Parameters

theDialog
A pointer to the dialog record.

theEvent
A pointer to the event record.

itemHit
The item number.

yourDataPtr
A pointer to the data received from your application, if any.

Return Value
Your `ModalFilterProc` callback returns a Boolean value that reports whether it handled the event. If your function returns a value of FALSE, QuickTime processes the event through its own filters. If your function returns a value of TRUE, QuickTime returns with no further action.

Discussion
The `ModalFilterProc` callback used with custom file dialogs requires the additional `yourDataPtr` parameter.

Declared In
`ImageCompression.h`

Data Types

GraphicsExportComponent

Represents a type used by the Graphics Import and Export API.

```
typedef ComponentInstance GraphicsExportComponent;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

GraphicsImportComponent

Represents a type used by the Graphics Import and Export API.

```
typedef ComponentInstance GraphicsImportComponent;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ImageCompression.h`

ModalFilterYDUPP

Represents a type used by the Graphics Import and Export API.

```
typedef STACK_UPP_TYPE(ModalFilterYDProcPtr) ModalFilterYDUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Dialogs.h`

Constants

GraphicsImportDoesDrawAllPixels Values

Constants passed to GraphicsImportDoesDrawAllPixels.

```
enum {
  graphicsImporterDrawsAllPixels = 0,
  graphicsImporterDoesntDrawAllPixels = 1,
  graphicsImporterDontKnowIfDrawAllPixels = 2
};
```

Declared In
`ImageCompression.h`

Graphics Importer Flags

Constants that represent the flags of graphics importers.

```
enum {
  graphicsImporterIsBaseImporter = 1L << 0,
  graphicsImporterCanValidateFile = 1L << 9,
  graphicsImporterSubTypeIsFileExtension = 1L << 12,
  graphicsImporterHasMIMEList   = 1L << 14,
  graphicsImporterUsesImageDecompressor = 1L << 23
};
enum {
  kGraphicsImporterDontDoGammaCorrection = 1L << 0,
  kGraphicsImporterTrustResolutionFromFile = 1L << 1,
  kGraphicsImporterEnableSubPixelPositioning = 1L << 2,
  kGraphicsImporterDontUseColorMatching = 1L << 3 /* set this flag (*before* calling
 GraphicsImportGetColorSyncProfile) if you do matching yourself */
};
```

Declared In
`ImageCompression.h`

GraphicsImportCreateCGImage Values

Constants passed to GraphicsImportCreateCGImage.

```
enum {
  kGraphicsImportCreateCGImageUsingCurrentSettings = 1L << 0
};
```

Declared In
`ImageCompression.h`

PNG Properties

Constants that represent the properties of PNGs.

```
enum {
  kQTPNGFilterPreference         = 'pngf', /* UInt32*/
  kQTPNGFilterBestForColorType  = 'bflt',
  kQTPNGFilterNone              = 0,
  kQTPNGFilterSub               = 1,
  kQTPNGFilterUp                = 2,
  kQTPNGFilterAverage           = 3,
  kQTPNGFilterPaeth             = 4,
  kQTPNGFilterAdaptivePerRow    = 'aflt',
  kQTPNGInterlaceStyle          = 'ilac', /* UInt32*/
  kQTPNGInterlaceNone           = 0,
  kQTPNGInterlaceAdam7          = 1
};
```

Constants

`kQTPNGFilterPreference`

UInt32.

Available in Mac OS X v10.0 and later.

Declared in `ImageCompression.h`.

`kQTPNGInterlaceStyle`

UInt32.

Available in Mac OS X v10.0 and later.

Declared in `ImageCompression.h`.

Declared In

`ImageCompression.h`

TIFF Properties

Constants that represent the properties of TIFFs.

```
enum {
  kQTTIFFCompressionMethod      = 'tifc', /* UInt32*/
  kQTTIFFCompression_None       = 1,
  kQTTIFFCompression_PackBits   = 32773L,
  kQTTIFFLittleEndian           = 'tife' /* UInt8 (boolean)*/
};
enum {
  kQTTIFFUserDataModelPixelScale = 0x7469830E, /* 3 DOUBLEs */
  kQTTIFFUserDataModelTransformation = 0x746985D8, /* 16 DOUBLEs */
  kQTTIFFUserDataModelTiepoint  = 0x74698482, /* n DOUBLEs */
  kQTTIFFUserDataGeoKeyDirectory = 0x746987AF, /* n SHORTs */
  kQTTIFFUserDataGeoDoubleParams = 0x746987B0, /* n DOUBLEs */
  kQTTIFFUserDataGeoAsciiParams = 0x746987B1, /* n ASCIIs */
  kQTTIFFUserDataIntergraphMatrix = 0x74698480 /* 16 or 17 DOUBLEs */
};
enum {
  kQTTIFFUserDataOrientation    = 0x74690112, /* 1 SHORT */
  kQTTIFFUserDataTransferFunction = 0x7469012D, /* n SHORTs */
  kQTTIFFUserDataWhitePoint     = 0x7469013E, /* 2 RATIONALs */
  kQTTIFFUserDataPrimaryChromaticities = 0x7469013F, /* 6 RATIONALs */
  kQTTIFFUserDataTransferRange  = 0x74690156, /* 6 SHORTs */
  kQTTIFFUserDataYCbCrPositioning = 0x74690213, /* 1 SHORT */
  kQTTIFFUserDataReferenceBlackWhite = 0x74690214 /* n LONGs */
};
```

Constants

`kQTTIFFCompressionMethod`

UInt32.

Available in Mac OS X v10.0 and later.

Declared in `ImageCompression.h`.

`kQTTIFFCompression_PackBits`

PackBits compression. This value is 32773L

Available in Mac OS X v10.0 and later.

Declared in `ImageCompression.h`.

`kQTTIFFLittleEndian`

UInt8 (Boolean).

Available in Mac OS X v10.0 and later.

Declared in `ImageCompression.h`.

`kQTTIFFUserDataModelPixelScale`

3 DOUBLEs.

Available in Mac OS X v10.2 and later.

Declared in `ImageCompression.h`.

`kQTTIFFUserDataModelTransformation`

16 DOUBLEs.

Available in Mac OS X v10.2 and later.

Declared in `ImageCompression.h`.

`kQTTIFFUserDataModelTiepoint`

N DOUBLEs.

Available in Mac OS X v10.2 and later.

Declared in `ImageCompression.h`.

`kQTTIFFUserDataGeoKeyDirectory`
N SHORTs.

Available in Mac OS X v10.2 and later.

Declared in `ImageCompression.h`.

`kQTTIFFUserDataGeoDoubleParams`
N DOUBLEs.

Available in Mac OS X v10.2 and later.

Declared in `ImageCompression.h`.

`kQTTIFFUserDataGeoAsciiParams`
N ASCIIs.

Available in Mac OS X v10.2 and later.

Declared in `ImageCompression.h`.

`kQTTIFFUserDataIntergraphMatrix`
16 or 17 DOUBLEs.

Available in Mac OS X v10.2 and later.

Declared in `ImageCompression.h`.

`kQTTIFFUserDataOrientation`
1 SHORT.

Available in Mac OS X v10.2 and later.

Declared in `ImageCompression.h`.

`kQTTIFFUserDataTransferFunction`
N SHORTs.

Available in Mac OS X v10.2 and later.

Declared in `ImageCompression.h`.

`kQTTIFFUserDataWhitePoint`
2 RATIONALs.

Available in Mac OS X v10.2 and later.

Declared in `ImageCompression.h`.

`kQTTIFFUserDataPrimaryChromaticities`
6 RATIONALs.

Available in Mac OS X v10.2 and later.

Declared in `ImageCompression.h`.

`kQTTIFFUserDataTransferRange`
6 SHORTs.

Available in Mac OS X v10.2 and later.

Declared in `ImageCompression.h`.

`kQTTIFFUserDataYCbCrPositioning`
1 SHORT.

Available in Mac OS X v10.2 and later.

Declared in `ImageCompression.h`.

Declared In
`ImageCompression.h`

Media Types and Media Handlers Reference

Framework:	Frameworks/QuickTime.framework
Declared in	MediaHandlers.h

Overview

QuickTime media handler components interpret and manipulate media types, such as sound, video, music, text, timecodes, and tweens.

Functions by Task

General Data Management

`MediaCompare` (page 1094)
Lets a media handler determine whether the Movie Toolbox should allow one track to be pasted into another.

`MediaGetMediaInfo` (page 1105)
Lets a derived media handler obtain the private data stored in its media.

`MediaGetName` (page 1106)
Returns the name of the media type.

`MediaGetNextStepTime` (page 1108)
Searches for the next forward or backward step time from the given media time.

`MediaGetOffscreenBufferSize` (page 1108)
Determines the dimensions of the offscreen buffer.

`MediaGetSampleDataPointer` (page 1111)
Allows a derived media handler to obtain a pointer to the sample data for a particular sample number, the size of that sample, and the index of the sample description associated with that sample.

`MediaGetVideoParam` (page 1118)
Retrieves the value of the brightness, contrast, hue, sharpness, saturation, black level, or white level of a video image.

`MediaGSetActiveSegment` (page 1120)
Informs your derived media handlers of the current active segment.

`MediaHasCharacteristic` (page 1122)
Called by Movie Toolbox with a specified characteristic to allow tracks to be identified by various attributes.

`MediaInvalidateRegion` (page 1128)
    Updates the invalidated display region the next time MediaIdle is called.

`MediaPreroll` (page 1132)
    Prepares a media handler for playback.

`MediaPutMediaInfo` (page 1133)
    Lets a derived media handler store proprietary information in its media.

`MediaReleaseSampleDataPointer` (page 1136)
    Balances calls to MediaGetSampleDataPointer to release allocated memory.

`MediaSampleDescriptionChanged` (page 1138)
    Informs a media handler that SetMediaSampleDescription has been called for a specified sample description.

`MediaSetActive` (page 1140)
    Enables and disables media.

`MediaSetHints` (page 1146)
    Implements the appropriate behavior for the various media hints such as scrub mode and high-quality mode.

`MediaSetMediaTimeScale` (page 1147)
    Informs a media handler that its media's time scale has been changed.

`MediaSetMovieTimeScale` (page 1148)
    Informs a media handler that the movie's time scale has been changed.

`MediaSetNonPrimarySourceData` (page 1149)
    Allows a media handler to support receiving media data from other media handlers.

`MediaSetRate` (page 1152)
    Sets a media's playback rate.

`MediaSetTrackInputMapReference` (page 1157)
    Provides a derived media handler with an updated input map.

`MediaSetVideoParam` (page 1159)
    Lets you dynamically adjust the brightness, contrast, hue, sharpness, saturation, black level, and white level of a video image.

`MediaTrackEdited` (page 1161)
    Informs a derived media handler about edits to its track.

`MediaTrackPropertyAtomChanged` (page 1162)
    Notifies the derived media handler whenever its media property atom has changed.

`MediaTrackReferencesChanged` (page 1162)
    Notifies the derived media handler whenever the track references in the movie change.

Managing Graphics Data

`MediaGetDrawingRgn` (page 1101)
    Specifies a portion of the screen that must be redrawn, defined in the movie's display coordinate system.

`MediaGetNextBoundsChange` (page 1107)
    Determines when a media causes a spatial change to a movie.

`MediaGetSrcRgn` (page 1115)
Specifies an irregular destination display region to the Movie Toolbox.

`MediaGetTrackOpaque` (page 1116)
Determines whether a media is transparent or opaque when displayed.

`MediaSetClip` (page 1141)
Specifies changes to a derived media handler's clipping region.

`MediaSetDimensions` (page 1142)
Informs a media handler when its media's spatial dimensions change.

`MediaSetGWorld` (page 1144)
Lets a derived media handler learn about changes to its media's graphic environment.

`MediaSetMatrix` (page 1147)
Tells a media handler about changes to either the movie matrix or the track matrix.

Managing Media Chunks

`MediaEmptyAllPurgeableChunks` (page 1097)
Force QuickTime to empty all purgeable media chunks in this application.

`MediaGetChunkManagementFlags` (page 1100)
Returns the current settings of the media chunk management flags.

`MediaGetPurgeableChunkMemoryAllowance` (page 1110)
Returns the current purgeable chunk memory allowance.

`MediaSetChunkManagementFlags` (page 1141)
Sets application-global flags that control media chunk management.

`MediaSetPurgeableChunkMemoryAllowance` (page 1152)
Sets the maximum amount of memory that QuickTime will allow purgeable chunks to occupy.

Managing Your Media Handler Component

`MediaGGetStatus` (page 1120)
Reports error conditions to the Movie Toolbox.

`MediaIdle` (page 1125)
Provides processing time to a derived media handler during movie playback.

`MediaInitialize` (page 1127)
Prepares a derived media handler component to provide access to its media.

Sound Media Handler Functions

`MediaGetSoundBalance` (page 1111)
Obtains the right/left sound balance of a track.

`MediaSetSoundBalance` (page 1154)
Sets the right/left sound balance of a track.

Supporting Keyboard Focus

MediaNavigateTargetRefCon (page 1131)
Locates the object for keyboard focus.

MediaRefConGetProperty (page 1135)
Returns the current media handler state based on the property type.

MediaRefConSetProperty (page 1136)
Sets a new media handler state based on the property type.

Video Media Handler Functions

MediaGetGraphicsMode (page 1104)
Obtains the graphics mode and blend color values currently in use by any media handler.

MediaSetGraphicsMode (page 1143)
Sets the graphics mode and blend color of any media handler.

Working With The Idle Manager

MediaGGetIdleManager (page 1119)
Retrieves an Idle Manager object from a derived media handler.

MediaGSetIdleManager (page 1121)
Lets a derived media handler report its idling needs.

Supporting Functions

CallComponentExecuteWiredAction (page 1093)
Undocumented

DisposePrePrerollCompleteUPP (page 1093)
Disposes of a PrePrerollCompleteUPP pointer.

MediaChangedNonPrimarySource (page 1094)
Informs a media handler of a change in the source of media data from another media handler.

MediaCurrentMediaQueuedData (page 1095)
Retrieves the timing of the current media in queued data.

MediaDisposeTargetRefCon (page 1096)
Disposes any resources allocated as part of calling MediaHitTestForTargetRefCon.

MediaDoIdleActions (page 1096)
Forces a media handler to perform its idle-time actions.

MediaEmptySampleCache (page 1097)
Deletes any sample data that the media handler has cached.

MediaEnterEmptyEdit (page 1098)
Undocumented

MediaFlushNonPrimarySourceData (page 1098)
Flushes data that a media handler gets from another media handler.

`MediaForceUpdate` (page 1099)
    Forces a media update.

`MediaGetActionsForQTEvent` (page 1099)
    Returns an event handler for your media handler.

`MediaGetClock` (page 1101)
    Gets the clock component associated with a media.

`MediaGetEffectiveSoundBalance` (page 1102)
    Gets the effective sound balance setting of a media handler.

`MediaGetEffectiveVolume` (page 1102)
    Gets the effective volume setting for a media handler.

`MediaGetErrorString` (page 1103)
    Undocumented

`MediaGetInvalidRegion` (page 1104)
    Gets the invalid region for a media handler's current display.

`MediaGetMediaLoadState` (page 1105)
    Queried by GetMovieLoadState to help determine a movie's load state.

`MediaGetPublicInfo` (page 1109)
    Undocumented

`MediaGetSoundBassAndTreble` (page 1112)
    Gets the bass and treble settings for a media handler.

`MediaGetSoundEqualizerBandLevels` (page 1113)
    Gets the sound equalizer band levels for a media handler.

`MediaGetSoundEqualizerBands` (page 1113)
    Gets the sound equalizer settings for a media handler.

`MediaGetSoundLevelMeterInfo` (page 1114)
    Gets the right and left sound level meter values for a media handler.

`MediaGetSoundLevelMeteringEnabled` (page 1114)
    Determines if a media handler's sound level metering capability is enabled.

`MediaGetSoundOutputComponent` (page 1115)
    Gets the sound output component associated with a media handler.

`MediaGetURLLink` (page 1117)
    Undocumented

`MediaGetUserPreferredCodecs` (page 1118)
    Retrieves the list of components last passed to the media handler by a call to MediaSetUserPreferredCodecs.

`MediaGSetVolume` (page 1122)
    Specifies changes to the sound volume setting.

`MediaHitTestForTargetRefCon` (page 1123)
    Locates an object for hit testing.

`MediaHitTestTargetRefCon` (page 1124)
    Detects if the mouse click and its release are in the same location and within the object.

`MediaMakeMediaTimeTable` (page 1129)
    Called by the base media handler to create a media time table.

MediaMCIsPlayerEvent (page 1130)
Undocumented

MediaPrePrerollBegin (page 1131)
Undocumented

MediaPrePrerollCancel (page 1132)
Cancels a media handler pre-preroll operation that was started by MediaPrePrerollBegin.

MediaQueueNonPrimarySourceData (page 1134)
Undocumented

MediaResolveTargetRefCon (page 1137)
Undocumented

MediaSampleDescriptionB2N (page 1138)
Undocumented

MediaSampleDescriptionN2B (page 1139)
Undocumented

MediaSetActionsCallback (page 1139)
Sets an ActionsProc callback for a media handler.

MediaSetDoMCActionCallback (page 1143)
Sets a DoMCActionProc callback for a media handler.

MediaSetHandlerCapabilities (page 1145)
Lets a derived media handler report its capabilities to the base media handler.

MediaSetPublicInfo (page 1151)
Undocumented

MediaSetScreenLock (page 1153)
Locks the display screen for a media handler.

MediaSetSoundBassAndTreble (page 1154)
Sets the bass and treble controls for a media handler.

MediaSetSoundEqualizerBands (page 1155)
Sets sound equalizer bands for a media handler.

MediaSetSoundLevelMeteringEnabled (page 1156)
Enables or disables sound level metering for a media handler.

MediaSetSoundLocalizationData (page 1156)
Supports 3D sound capabilities in a media handler that plays sound.

MediaSetSoundOutputComponent (page 1157)
Sets the sound output component for a media handler.

MediaSetUserPreferredCodecs (page 1158)
Requests that a media handler favor specified codec components when selecting components with which to play media.

MediaTargetRefConsEqual (page 1160)
Undocumented

MediaTimeBaseChanged (page 1161)
Undocumented

MediaVideoOutputChanged (page 1163)
Undocumented

NewPrePrerollCompleteUPP (page 1163)
: Allocates a Universal Procedure Pointer for the PrePrerollCompleteProc callback.

Functions

CallComponentExecuteWiredAction

Undocumented

```
ComponentResult CallComponentExecuteWiredAction (
   ComponentInstance ci,
   QTAtomContainer actionContainer,
   QTAtom actionAtom,
   QTCustomActionTargetPtr target,
   QTEventRecordPtr event
);
```

Parameters

ci
: A component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

actionContainer
: A QT atom container that contains the action atom.

actionAtom
: The action atom for this wired action.

target
: A pointer to a `QTCustomActionTargetRecord` structure.

event
: A pointer to a `QTEventRecord` structure.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MediaHandlers.h`

DisposePrePrerollCompleteUPP

Disposes of a PrePrerollCompleteUPP pointer.

```
void DisposePrePrerollCompleteUPP (
   PrePrerollCompleteUPP userUPP
);
```

Parameters

userUPP

A `PrePrerollCompleteUPP` pointer. See `Universal Procedure Pointers`.

Return Value

You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes

Introduced in QuickTime 4.1.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`MediaHandlers.h`

MediaChangedNonPrimarySource

Informs a media handler of a change in the source of media data from another media handler.

```
ComponentResult MediaChangedNonPrimarySource (
   MediaHandler mh,
   long inputIndex
);
```

Parameters

mh

A reference to a media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

inputIndex

The ID of the entry in the media's input map to which the changed data corresponds.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MediaHandlers.h`

MediaCompare

Lets a media handler determine whether the Movie Toolbox should allow one track to be pasted into another.

```
ComponentResult MediaCompare (
   MediaHandler mh,
   Boolean *isOK,
   Media srcMedia,
   ComponentInstance srcMediaComponent
);
```

Parameters

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

isOK

A pointer to a Boolean value. Your media handler must set this value to TRUE if the source media and the media associated with the media handler have equivalent media settings, so that pasting the two together would cause no media information loss.

srcMedia

The source media for this operation.

srcMediaComponent

The source media component for this operation.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MediaHandlers.h`

MediaCurrentMediaQueuedData

Retrieves the timing of the current media in queued data.

```
ComponentResult MediaCurrentMediaQueuedData (
   MediaHandler mh,
   long *milliSecs
);
```

Parameters

mh

A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

milliSecs

A pointer to the number of milliseconds to the current data.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MediaHandlers.h`

MediaDisposeTargetRefCon

Disposes any resources allocated as part of calling MediaHitTestForTargetRefCon.

```
ComponentResult MediaDisposeTargetRefCon (
   MediaHandler mh,
   long targetRefCon
);
```

Parameters

mh
A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

targetRefCon
A reference constant set by the media handler in a call to `MediaHitTestForTargetRefCon` (page 1123).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MediaHandlers.h`

MediaDoIdleActions

Forces a media handler to perform its idle-time actions.

```
ComponentResult MediaDoIdleActions (
   MediaHandler mh
);
```

Parameters

mh
A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
SurfaceVertexProgram

Declared In
`MediaHandlers.h`

MediaEmptyAllPurgeableChunks

Force QuickTime to empty all purgeable media chunks in this application.

```
ComponentResult MediaEmptyAllPurgeableChunks (
   MediaHandler mh
);
```

Parameters

`mh`
: The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 6. Can be used only with Mac OS X 10.1 and later.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`MediaHandlers.h`

MediaEmptySampleCache

Deletes any sample data that the media handler has cached.

```
ComponentResult MediaEmptySampleCache (
   MediaHandler mh,
   long sampleNum,
   long sampleCount
);
```

Parameters

`mh`
: A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

`sampleNum`
: The ID of the first sample to delete.

`sampleCount`
: The number of samples to delete. Passing -1 means delete `sampleNum` and all samples after it.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This is an optional media handler function. Most developers will not need to call it.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MediaHandlers.h`

MediaEnterEmptyEdit

Undocumented

```
ComponentResult MediaEnterEmptyEdit (
   MediaHandler mh
);
```

Parameters

mh
  A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MediaHandlers.h`

MediaFlushNonPrimarySourceData

Flushes data that a media handler gets from another media handler.

```
ComponentResult MediaFlushNonPrimarySourceData (
   MediaHandler mh,
   long inputIndex
);
```

Parameters

mh
  A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

inputIndex
  The ID of the entry in the media's input map to which the flushed data corresponds.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MediaHandlers.h`

MediaForceUpdate

Forces a media update.

```
ComponentResult MediaForceUpdate (
   MediaHandler mh,
   long forceUpdateFlags
);
```

Parameters

mh
A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

forceUpdateFlags
Flags (see below) that define the update to be forced. See these constants:
`forceUpdateRedraw`
`forceUpdateNewBuffer`

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MediaHandlers.h`

MediaGetActionsForQTEvent

Returns an event handler for your media handler.

```
ComponentResult MediaGetActionsForQTEvent (
   MediaHandler mh,
   QTEventRecordPtr event,
   long targetRefCon,
   QTAtomContainer *container,
   QTAtom *atom
);
```

Parameters

mh
A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

event
A pointer to a `QTEventRecord` structure.

targetRefCon

A reference constant set by the media handler in `MediaHitTestForTargetRefCon` (page 1123).

container

An atom container that you can pass back to the standard controller used for implementing sprite actions.

atom

An atom you can pass back to the standard controller used for implementing sprite actions.

Return Value

See `Error Codes`. Returns `qtEventWasHandledErr` if the event was handled by the media handler. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MediaHandlers.h`

MediaGetChunkManagementFlags

Returns the current settings of the media chunk management flags.

```
ComponentResult MediaGetChunkManagementFlags (
   MediaHandler mh,
   UInt32 *flags
);
```

Parameters

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

flags

A pointer to the constants (see below) that were set by a previous call to `MediaSetChunkManagementFlags` (page 1141). See these constants:

`kEmptyPurgableChunksOverAllowance`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Do not call this function under QuickTime 5. It could cause a crash.

Version Notes

Introduced in QuickTime 6. Can be used only with Mac OS X 10.1 and later.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`MediaHandlers.h`

MediaGetClock

Gets the clock component associated with a media.

```
ComponentResult MediaGetClock (
   MediaHandler mh,
   ComponentInstance *clock
);
```

Parameters

mh

A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

clock

A pointer to a clock component.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MediaHandlers.h`

MediaGetDrawingRgn

Specifies a portion of the screen that must be redrawn, defined in the movie's display coordinate system.

```
ComponentResult MediaGetDrawingRgn (
   MediaHandler mh,
   RgnHandle *partialRgn
);
```

Parameters

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

partialRgn

A pointer to a handle to a `MacRegion` structure that defines the screen region to be redrawn, using the movie's display coordinate system. Note that your component is responsible for disposing of this region once drawing is complete. Since the base media handler will use this region during redrawing, it is best to dispose of it when your component is closed.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The Movie Toolbox calls this function in order to determine what part of the screen needs to be redrawn. By default, `theMovie` Toolbox redraws the entire region that belongs to your component. If your component determines that only a portion of the screen has changed, and has indicated this to the toolbox by setting the `mPartialDraw` flag to 1 in the `flagsOut` parameter of the `MediaIdle` (page 1125) function, the toolbox calls your component's `MediaGetDrawingRgn` (page 1101) function. Your component returns a region that defines the changed portion of the track's display region.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MediaHandlers.h`

MediaGetEffectiveSoundBalance

Gets the effective sound balance setting of a media handler.

```
ComponentResult MediaGetEffectiveSoundBalance (
   MediaHandler mh,
   short *balance
);
```

Parameters

mh

A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

balance

A pointer to an integer. The Movie Toolbox returns the current balance setting of the media handler as a 16-bit, fixed-point value. The high-order 8 bits contain the integer part of the value; the low-order 8 bits contain the fractional part. Valid balance values range from -1.0 to 1.0. Negative values emphasize the left sound channel, and positive values emphasize the right sound channel; a value of 0 specifies neutral balance.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MediaHandlers.h`

MediaGetEffectiveVolume

Gets the effective volume setting for a media handler.

```
ComponentResult MediaGetEffectiveVolume (
   MediaHandler mh,
   short *volume
);
```

Parameters

mh

A media handler. You can obtain this reference from GetMediaHandler (page 1577).

volume

The media's current volume setting. This value is represented as a 16-bit, fixed-point number. The high-order 8 bits contain the integer portion; the low-order 8 bits contain the fractional part. Volume values range from -1.0 to 1.0. Negative values play no sound but preserve the absolute value of the volume setting.

Return Value
See Error Codes. Returns noErr if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
MediaHandlers.h

MediaGetErrorString

Undocumented

```
ComponentResult MediaGetErrorString (
   MediaHandler mh,
   ComponentResult theError,
   Str255 errorString
);
```

Parameters

mh

A media handler. You can obtain this reference from GetMediaHandler (page 1577).

theError

An error identifier; see Error Codes.

errorString

A text string that describes the error.

Return Value
See Error Codes. Returns noErr if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MediaHandlers.h`

MediaGetGraphicsMode

Obtains the graphics mode and blend color values currently in use by any media handler.

```
ComponentResult MediaGetGraphicsMode (
   MediaHandler mh,
   long *mode,
   RGBColor *opColor
);
```

Parameters

mh
: The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

mode
: A pointer to a long integer. The media handler returns the graphics mode currently in use by the media handler; see `Graphics Transfer Modes`.

opColor
: A pointer to an `RGBColor` structure. The Movie Toolbox returns the color currently in use by the media handler. This is the blend value for blends and the transparent color for transparent operations. The toolbox supplies this value to QuickDraw when you draw in `addPin`, `subPin`, `blend`, `transparent`, or `graphicsModeStraightAlphaBlend` mode.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MediaHandlers.h`

MediaGetInvalidRegion

Gets the invalid region for a media handler's current display.

```
ComponentResult MediaGetInvalidRegion (
   MediaHandler mh,
   RgnHandle rgn
);
```

Parameters

mh
: A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

rgn
: A handle to a `MacRegion` structure that defines an invalid region.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MediaHandlers.h`

MediaGetMediaInfo

Lets a derived media handler obtain the private data stored in its media.

```
ComponentResult MediaGetMediaInfo (
   MediaHandler mh,
   Handle h
);
```

Parameters

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

h

A handle to storage containing your media handler's proprietary information. Your media handler creates this private data when the Movie Toolbox calls your `MediaPutMediaInfo` (page 1133) function. Do not dispose of this handle; it is owned by the Movie Toolbox.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Your derived media handler should support this function if you store private data in your media.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MediaHandlers.h`

MediaGetMediaLoadState

Queried by GetMovieLoadState to help determine a movie's load state.

```
ComponentResult MediaGetMediaLoadState (
   MediaHandler mh,
   long *mediaLoadState
);
```

Parameters

mh

A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

mediaLoadState

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

`GetMovieLoadState` (page 1358) queries any idling importers associated with a movie, checks if the movie is fast starting, and queries media handlers. The minimum load state of all of these is then considered to be the load state of the movie.

Version Notes

Introduced in QuickTime 4.1.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MediaHandlers.h`

MediaGetName

Returns the name of the media type.

```
ComponentResult MediaGetName (
   MediaHandler mh,
   Str255 name,
   long requestedLanguage,
   long *actualLanguage
);
```

Parameters

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

name

The name of the `media` type; for example, the video media handler returns the string `'video'`.

requestedLanguage

The language in which you want the name returned; see `Localization Codes`.

actualLanguage

A pointer to the actual language in which the name is returned; see `Localization Codes`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
MakeEffectMovie
qteffects.win
qtgraphics.win
qtwiredactions
vrbackbuffer.win

Declared In
`MediaHandlers.h`

MediaGetNextBoundsChange

Determines when a media causes a spatial change to a movie.

```
ComponentResult MediaGetNextBoundsChange (
   MediaHandler mh,
   TimeValue *when
);
```

Parameters

mh
: The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

when
: A pointer to a movie time value, which your media handler must set. Be sure to use the movie's time base. Use the current effective rate to determine the direction your media is playing. Set this value to -1 if there are no more changes in the specified direction.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Your derived media handler should support this function if you change the shape of your media's spatial representation during playback. The Movie Toolbox calls this function only if you have set the `handlerHasSpatial` flag to 1 in the `flags` parameter of `MediaSetHandlerCapabilities` (page 1145).

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MediaHandlers.h`

MediaGetNextStepTime

Searches for the next forward or backward step time from the given media time.

```
ComponentResult MediaGetNextStepTime (
   MediaHandler mh,
   short flags,
   TimeValue mediaTimeIn,
   TimeValue *mediaTimeOut,
   Fixed rate
);
```

Parameters

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

flags

Flags (see below) that specify search parameters. See these constants:

`nextTimeStep`

mediaTimeIn

A time value that establishes the starting point for the search. This time value is in the media's time scale.

mediaTimeOut

The step time (the time of the next frame) calculated by the media handler. The media handler should return 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.

rate

The search direction. Negative values search backward from the starting point specified in the `mediaTimeIn` parameter. Other values cause a forward search.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function allows a derived media handler to return the next step time from the specified media time. The mechanism in QuickTime used for stepping backwards and forwards a frame at a time are the interesting time calls: `GetMovieNextInterestingTime` (page 1359), `GetTrackNextInterestingTime` (page 1373), and `GetMediaNextInterestingTime` (page 1346).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MediaHandlers.h`

MediaGetOffscreenBufferSize

Determines the dimensions of the offscreen buffer.

```
ComponentResult MediaGetOffscreenBufferSize (
   MediaHandler mh,
   Rect *bounds,
   short depth,
   CTabHandle ctab
);
```

Parameters

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

bounds

A `Rect` structure that defines the boundaries of your offscreen buffer.

depth

The depth of the offscreen.

ctab

A handle to the `ColorTable` structure associated with the offscreen buffer.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Before the base media handler allocates an offscreen buffer for your derived media handler, it calls this function. The depth and color table used for the buffer are also passed. When this function is called, the `bounds` parameter specifies the size that the base media handler intends to use for your offscreen buffer. You can modify this as appropriate before returning. This capability is useful if your media handler can draw only at particular sizes. It is also useful for implementing antialiased drawing; you can request a buffer that is larger than your destination area and have the base media handler scale the image down for you.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MediaHandlers.h`

MediaGetPublicInfo

Undocumented

```
ComponentResult MediaGetPublicInfo (
   MediaHandler mh,
   OSType infoSelector,
   void *infoDataPtr,
   Size *ioDataSize
);
```

Parameters

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

infoSelector
: *Undocumented*

infoDataPtr
: *Undocumented*

ioDataSize
: *Undocumented*

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qtskins

qtskins.win

Declared In
`MediaHandlers.h`

MediaGetPurgeableChunkMemoryAllowance

Returns the current purgeable chunk memory allowance.

```
ComponentResult MediaGetPurgeableChunkMemoryAllowance (
   MediaHandler mh,
   Size *allowance
);
```

Parameters

mh
: The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

allowance
: A pointer to the allowance in bytes.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 6. Can be used only with Mac OS X 10.1 and later.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`MediaHandlers.h`

MediaGetSampleDataPointer

Allows a derived media handler to obtain a pointer to the sample data for a particular sample number, the size of that sample, and the index of the sample description associated with that sample.

```
ComponentResult MediaGetSampleDataPointer (
   MediaHandler mh,
   long sampleNum,
   Ptr *dataPtr,
   long *dataSize,
   long *sampleDescIndex
);
```

Parameters

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

sampleNum

The number of the sample that is to be loaded.

dataPtr

A pointer to a pointer to receive the address of the loaded sample data.

dataSize

A pointer to a field that is to receive the size, in bytes, of the sample.

sampleDescIndex

A pointer to a long integer. This function returns an index value to the sample description that corresponds to the returned sample data. If you do not want this information, set the parameter to `NIL`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function returns a pointer to the data for a particular sample number from a movie data file. It provides access to the base media handler's caching services for sample data. It is a service provided by the base media handler for its clients.

Special Considerations

Each call to this function must be balanced by a call to `MediaReleaseSampleDataPointer` (page 1136) or the memory will not be released. This function generally provides better overall performance than `GetMediaSample` (page 1583).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MediaHandlers.h`

MediaGetSoundBalance

Obtains the right/left sound balance of a track.

```
ComponentResult MediaGetSoundBalance (
   MediaHandler mh,
   short *balance
);
```

Parameters

mh

A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

balance

On return, a pointer to the balance setting for the media handler's track. The balance setting is a signed 16-bit integer that controls the relative volume of the left and right sound channels. A value of 0 sets the balance to neutral. Positive values shift the balance to the right channel, negative values to the left channel. The valid range is 127 (right channel only) to -128 (left channel only).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Use this function to get the audio balance on a per-track basis. It works with both mono and stereo sources. The actual volume of the audio is not changed by this setting, only its relative distribution. This function operates on sound tracks, music tracks, and Flash tracks containing audio. It may operate on other audio track types as well; if the the chosen media handler does not support this function, it returns `badComponentSelector`. To obtain the media handler for a track, call `GetTrackMedia` (page 1612) and then `GetMediaHandler` (page 1577).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MediaHandlers.h`

MediaGetSoundBassAndTreble

Gets the bass and treble settings for a media handler.

```
ComponentResult MediaGetSoundBassAndTreble (
   MediaHandler mh,
   short *bass,
   short *treble
);
```

Parameters

mh

A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

bass

Undocumented

treble

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MediaHandlers.h`

MediaGetSoundEqualizerBandLevels

Gets the sound equalizer band levels for a media handler.

```
ComponentResult MediaGetSoundEqualizerBandLevels (
   MediaHandler mh,
   UInt8 *bandLevels
);
```

Parameters

mh
A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

bandLevels
Undocumented

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
sndequalizer
SurfaceVertexProgram

Declared In
`MediaHandlers.h`

MediaGetSoundEqualizerBands

Gets the sound equalizer settings for a media handler.

```
ComponentResult MediaGetSoundEqualizerBands (
   MediaHandler mh,
   MediaEQSpectrumBandsRecordPtr spectrumInfo
);
```

Parameters

mh
A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

spectrumInfo

A pointer to a `MediaEQSpectrumBandsRecord` structure.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MediaHandlers.h`

MediaGetSoundLevelMeterInfo

Gets the right and left sound level meter values for a media handler.

```
ComponentResult MediaGetSoundLevelMeterInfo (
   MediaHandler mh,
   LevelMeterInfoPtr levelInfo
);
```

Parameters

mh

A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

levelInfo

A pointer to a `LevelMeterInfo` structure.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MediaHandlers.h`

MediaGetSoundLevelMeteringEnabled

Determines if a media handler's sound level metering capability is enabled.

```
ComponentResult MediaGetSoundLevelMeteringEnabled (
   MediaHandler mh,
   Boolean *enabled
);
```

Parameters

mh

A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

enabled

A pointer to a Boolean; it is TRUE if sound level metering is enabled, FALSE otherwise.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MediaHandlers.h`

MediaGetSoundOutputComponent

Gets the sound output component associated with a media handler.

```
ComponentResult MediaGetSoundOutputComponent (
   MediaHandler mh,
   Component *outputComponent
);
```

Parameters

mh

A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

outputComponent

An instance of a sound output component.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MediaHandlers.h`

MediaGetSrcRgn

Specifies an irregular destination display region to the Movie Toolbox.

```
ComponentResult MediaGetSrcRgn (
   MediaHandler mh,
   RgnHandle rgn,
   TimeValue atMediaTime
);
```

Parameters

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

rgn

A handle to a `MacRegion` structure. When the Movie Toolbox calls your function, this region is initialized to the track's boundary rectangle, which is defined by the `width` and height fields in the `GetMovieCompleteParams` structure that you obtain when the Movie Toolbox calls your `MediaInitialize` (page 1127) function. Your media handler may then alter this region as appropriate, so that it corresponds to the boundaries of your media's display image. Note that this region is in the track's coordinate system, not the movie's. Do not dispose of this region; it is owned by the Movie Toolbox.

atMediaTime

The time value at which the Movie Toolbox wants to know what the source region is.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Your derived media handler should support this function if your media does not completely fill the track rectangle during playback. The Movie Toolbox calls this function only if you have set the `handlerHasSpatial` flag to 1 in the `flags` parameter of the `MediaSetHandlerCapabilities` (page 1145) function.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MediaHandlers.h`

MediaGetTrackOpaque

Determines whether a media is transparent or opaque when displayed.

```
ComponentResult MediaGetTrackOpaque (
   MediaHandler mh,
   Boolean *trackIsOpaque
);
```

Parameters

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

trackIsOpaque

A pointer to a Boolean value. Your media handler must set this value to TRUE if your media is semitransparent (that is, you draw in blend mode); otherwise, leave the flag unchanged.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Your derived media handler should support this function if your media is semitransparent when displayed or if you handle display transfer modes. The Movie Toolbox calls this function only if you have set the `handlerHasSpatial` or `handlerCanTransferMode` flag to 1 in the `flags` parameter of the `MediaSetHandlerCapabilities` (page 1145) function.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MediaHandlers.h`

MediaGetURLLink

Undocumented

```
ComponentResult MediaGetURLLink (
   MediaHandler mh,
   Point displayWhere,
   Handle *urlLink
);
```

Parameters

mh

A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

displayWhere

Undocumented

urlLink

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MediaHandlers.h`

MediaGetUserPreferredCodecs

Retrieves the list of components last passed to the media handler by a call to MediaSetUserPreferredCodecs.

```
ComponentResult MediaGetUserPreferredCodecs (
   MediaHandler mh,
   CodecComponentHandle *userPreferredCodecs
);
```

Parameters

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

userPreferredCodecs

A pointer to a handle containing component identifiers. If the media handler currently has a preferred component list, it will copy that list into a new handle and store the new handle in this variable. If the media handler does not currently have a preferred component list, it will store `NIL` in this variable. The caller must dispose of this handle.

Return Value

See `Error Codes`. Returns `badComponentSelector` if the media handler component does not support this call. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MediaHandlers.h`

MediaGetVideoParam

Retrieves the value of the brightness, contrast, hue, sharpness, saturation, black level, or white level of a video image.

```
ComponentResult MediaGetVideoParam (
   MediaHandler mh,
   long whichParam,
   unsigned short *value
);
```

Parameters

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

whichParam

A constant (see below) that specifies the video parameter whose value you want to retrieve. See these constants:

`kMediaVideoParamBrightness`

`kMediaVideoParamContrast`

`kMediaVideoParamHue`

`kMediaVideoParamSharpness`

`kMediaVideoParamSaturation`

`kMediaVideoParamBlackLevel`

`kMediaVideoParamWhiteLevel`

value

The actual value of the `requested` video parameter. The meaning of the values vary depending on the implementation.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function and `MediaSetVideoParam` (page 1159) are currently used by the MPEG media handler.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MediaHandlers.h`

MediaGGetIdleManager

Retrieves an Idle Manager object from a derived media handler.

```
ComponentResult MediaGGetIdleManager (
   MediaHandler mh,
   IdleManager *pim
);
```

Parameters

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

pim

A pointer to a pointer to an opaque data structure that belongs to the Mac OS Idle Manager. You can get the pointer this parameter points to by calling `QTIdleManagerOpen` (page 273).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This routine must be implemented by a derived media handler if the handler needs to report its idling requirements.

Version Notes
Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`MediaHandlers.h`

MediaGGetStatus

Reports error conditions to the Movie Toolbox.

```
ComponentResult MediaGGetStatus (
   MediaHandler mh,
   ComponentResult *statusErr
);
```

Parameters

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

statusErr

A pointer to a component result field. If you have error information that you would like to report to the Movie Toolbox, place an appropriate result code into the field referred to by this pointer. See `Error Codes`.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Your derived media handler should support this function if you anticipate that you may encounter an error when playing your media. Because these errors may include such conditions as low memory or missing hardware, you should only rarely create a derived media handler that does not support this function. If your media handler does not support this function, the base media handler always sets the returned result code to `noErr`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MediaHandlers.h`

MediaGSetActiveSegment

Informs your derived media handlers of the current active segment.

```
ComponentResult MediaGSetActiveSegment (
   MediaHandler mh,
   TimeValue activeStart,
   TimeValue activeDuration
);
```

Parameters

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

activeStart

The starting time of the active segment to play. This time value is expressed in your movie's time scale.

activeDuration

A time value that specifies the duration of the active segment. This value is expressed in the movie's time scale.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Using `SetMovieActiveSegment` (page 285), an application can limit the time segment of the movie that will be used for play back. Derived media handlers are given the values for the active segment when this function is called by the Movie Toolbox. Active segment information is usually only needed by media handlers that perform their own scheduling.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MediaHandlers.h`

MediaGSetIdleManager

Lets a derived media handler report its idling needs.

```
ComponentResult MediaGSetIdleManager (
   MediaHandler mh,
   IdleManager im
);
```

Parameters

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

im

A pointer to an opaque data structure that belongs to the Mac OS Idle Manager. You get this pointer by calling `QTIdleManagerOpen` (page 273).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This routine must be implemented by a derived media handler if the handler needs to retrieve and report its idling requirements.

Version Notes

Introduced in QuickTime 6.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`MediaHandlers.h`

MediaGSetVolume

Specifies changes to the sound volume setting.

```
ComponentResult MediaGSetVolume (
   MediaHandler mh,
   short volume
);
```

Parameters

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

volume

The media's current volume setting. This value is represented as a 16-bit, fixed-point number. The high-order 8 bits contain the integer portion; the low-order 8 bits contain the fractional part. Volume values range from -1.0 to 1.0. Negative values play no sound but preserve the absolute value of the volume setting. The Movie Toolbox scales your media's volume in light of the track's and movie's volume settings, but it does not take into account the system speaker volume setting. This value is appropriate for use with the Sound Manager.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Your derived media handler should support this function if it can play sounds.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MediaHandlers.h`

MediaHasCharacteristic

Called by Movie Toolbox with a specified characteristic to allow tracks to be identified by various attributes.

```
ComponentResult MediaHasCharacteristic (
   MediaHandler mh,
   OSType characteristic,
   Boolean *hasIt
);
```

Parameters

mh

The Movie Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

characteristic

A constant that specifies the attribute of a track. Examples of characteristics that are currently defined are the constants `VisualMediaCharacteristic` and `AudioMediaCharacteristic`.

hasIt

A pointer to a Boolean value that specifies whether the track has the attribute specified in the characteristic parameter. Set this value to TRUE if the attribute applies to your media handler; otherwise, set this value to FALSE.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

You should implement this function for any media handler that has characteristics in addition to spatial ones. If you have set the `handlerHasSpatial` capabilities flag, the base media handler automatically handles the `VisualMediaCharacteristic` constant for you.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

ElectricImageComponent

ElectricImageComponent.win

Declared In

`MediaHandlers.h`

MediaHitTestForTargetRefCon

Locates an object for hit testing.

```
ComponentResult MediaHitTestForTargetRefCon (
   MediaHandler mh,
   long flags,
   Point loc,
   long *targetRefCon
);
```

Parameters

mh

A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

flags

Flags (see below) that define the hit. The `mHitTestImage` and `mHitTestInvisible` flags are set by the Standard Controller before this call is made. See these constants:

`mHitTestBounds`

`mHitTestImage`

`mHitTestInvisible`

`mHitTestIsClick`

loc

The location of the mouse.

targetRefCon

Returns a reference constant representing an object you're interested in. If this reference constant is not 0, your media handler will receive calls to `MediaGetActionsForQTEvent` (page 1099).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MediaHandlers.h`

MediaHitTestTargetRefCon

Detects if the mouse click and its release are in the same location and within the object.

```
ComponentResult MediaHitTestTargetRefCon (
   MediaHandler mh,
   long targetRefCon,
   long flags,
   Point loc,
   Boolean *wasHit
);
```

Parameters

mh

A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

targetRefCon

A reference constant set by the media handler in a call to `MediaHitTestForTargetRefCon` (page 1123).

flags

Flags (see below) that define the hit. See these constants:

`mHitTestBounds`

`mHitTestImage`

`mHitTestInvisible`

`mHitTestIsClick`

loc
: The location of the mouse.

wasHit
: A pointer to a Boolean; it is TRUE if there was a hit, FALSE otherwise.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function is called after `MediaGetActionsForQTEvent` (page 1099) if a reference constant was set in `MediaHitTestForTargetRefCon` (page 1123).

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MediaHandlers.h`

MediaIdle

Provides processing time to a derived media handler during movie playback.

```
ComponentResult MediaIdle (
   MediaHandler mh,
   TimeValue atMediaTime,
   long flagsIn,
   long *flagsOut,
   const TimeRecord *movieTime
);
```

Parameters

mh
: The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

atMediaTime
: The current time, in your media's time base. You can use this value to obtain the appropriate samples and sample descriptions from your media (using the Movie Toolbox's `GetMediaSample` (page 1583) function). Your media handler may then work with the sample data and descriptions as appropriate.

flagsIn

Contains flags (see below) that indicate what the Movie Toolbox wants your media handler to do. These flags are applicable only to media handlers that perform their own scheduling. The toolbox may use none, or it may set one or more flag to 1. Your handler should examine the `flagsIn` parameter each time the Movie Toolbox calls its `MediaIdle` function. The flags in this parameter indicate the actions that your handler may perform. In addition, when you return from your `MediaIdle` function, you should report what you did using the `flagsOut` parameter. You tell the base media handler that you perform your own scheduling by setting the `handlerNoScheduler` flag to 1 in the `flags` parameter of the `MediaSetHandlerCapabilities` (page 1145) function. See these constants:

`mMustDraw`

`mAtEnd`

`mPartialDraw`

`mPreflightDraw`

flagsOut

Flags (see below) that are contained in a pointer to a long integer that your media handler uses to indicate to the Movie Toolbox what the handler did. You must always set the values of these flags appropriately. See these constants:

`mDidDraw`

`mNeedsToDraw`

movieTime

A pointer to the movie time value corresponding to the `atMediaTime` parameter. Note that this may differ from the current value returned by `GetMovieTime` (page 223).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

From time to time, your derived media handler component may determine that only a portion of the available drawing area needs to be redrawn. You can signal that condition to the base media handler component by setting the `mPartialDraw` flag to 1 in the flags your component returns to the Movie Toolbox from your `MediaIdle` function. You return these flags using the `flagsOut` parameter. Whenever you set this flag to 1, the Movie Toolbox calls your component's `MediaGetDrawingRgn` (page 1101) function in order to determine the portion of the image that needs to be redrawn.

As an example, consider a full-screen animation. Only rarely is the entire image in motion. Typically, only a small portion of the screen image moves. By using partial redrawing, you can significantly improve the playback performance of such a movie.

Your derived media handler should support this function if you need to do work during movie playback. If you set the `handlerNoIdle` flag to 1 in the `flags` parameter of `MediaSetHandlerCapabilities` (page 1145), the Movie Toolbox does not call your `MediaIdle` function. If you encounter an error, save the result code. The Movie Toolbox polls you for status information using `MediaGGetStatus` (page 1120).

Special Considerations

If your media handler changes any of the settings of the movie's graphics port or graphics world, be sure to restore the original settings before you exit. In addition, note that you may be drawing into a black-and-white graphics port. Finally, be aware that the Movie Toolbox also uses this function to obtain data for QuickDraw pictures. Therefore, if your media handler does not use QuickDraw when drawing to the screen, be sure to examine the `picSave` field in the graphics port so that you can detect when the Movie Toolbox wants to save an image. Your media handler is then responsible for performing the appropriate display processing.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MediaHandlers.h`

MediaInitialize

Prepares a derived media handler component to provide access to its media.

```
ComponentResult MediaInitialize (
   MediaHandler mh,
   GetMovieCompleteParams *gmc
);
```

Parameters

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

gmc

A pointer to a `GetMovieCompleteParams` structure. You can obtain information about the current media from this structure. You should copy any values you need to save into your derived media handler's local data area. Because this data structure is owned by the Movie Toolbox, you do not need to worry about disposing of any of the data in it.

Return Value
See `Error Codes`. Returns `noErr` if there is no error. If you return an error, the Movie Toolbox disables the track that uses your media. In cases where your media has just been created, the Movie Toolbox immediately disposes of your media.

Discussion
This function gives your media handler an opportunity to get ready to support the Movie Toolbox. As part of these preparations, your derived media handler should report its capabilities to the base media handler by calling `MediaSetHandlerCapabilities` (page 1145). You may choose to examine the data in the `GetMovieCompleteParams` structure; you may also save values from this structure. If you save references to structures (such as the matte pixel map), do not dispose of the memory associated with these structures. The Movie Toolbox owns these structures.

Note that the Movie Toolbox may call other functions supported by your media handler before it calls your `MediaInitialize` function. In particular, it may call your `MediaGetMediaInfo` (page 1105) and `MediaPutMediaInfo` (page 1133) functions. However, before the Movie Toolbox tries to do anything with the data in your media, it will call your `MediaInitialize` function. The Movie Toolbox loads the movie's data using functions that are supported by the base media handler; your media handler does not have to support those functions.

Special Considerations

All derived media handlers should support this function. In addition, if your media handler saves values from the `GetMovieCompleteParams` structure that may change, be sure to support the corresponding functions that allow the Movie Toolbox to report changes to your media handler. For example, if your handler saves the movie time scale from the `movieScale` field, you should also support the `MediaSetMovieTimeScale` (page 1148) function.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
SurfaceVertexProgram

Declared In
`MediaHandlers.h`

MediaInvalidateRegion

Updates the invalidated display region the next time MediaIdle is called.

```
ComponentResult MediaInvalidateRegion (
   MediaHandler mh,
   RgnHandle invalRgn
);
```

Parameters

mh
: The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

invalRgn
: A handle to a region that has been invalidated. Your media handler should not dispose or modify this region. The `invalRgn` parameter is never `NIL`.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function is called by the Movie Toolbox when `UpdateMovie` (page 347) or `InvalidateMovieRegion` (page 241) is called with a region that intersects your media's track. Derived media handlers need to implement `MediaInvalidateRegion` only if they can perform efficient updates on a portion of their display area.

If a media handler implements this function, it is responsible for ensuring that the appropriate areas of the screen are updated on the next call to `MediaIdle` (page 1125). If a media handler does not implement this function, the base media handler sets the `mMustDraw` flag the next time `MediaIdle` is called.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MediaHandlers.h`

MediaMakeMediaTimeTable

Called by the base media handler to create a media time table.

```
ComponentResult MediaMakeMediaTimeTable (
   MediaHandler mh,
   long **offsets,
   TimeValue startTime,
   TimeValue endTime,
   TimeValue timeIncrement,
   short firstDataRefIndex,
   short lastDataRefIndex,
   long *retDataRefSkew
);
```

Parameters

mh

The media handler to create the time table. You can obtain this reference from `GetMediaHandler` (page 1577).

offsets

A handle to an unlocked relocatable memory block that is allocated by an application or other software when it calls `QTMovieNeedsTimeTable` (page 1450), `GetMaxLoadedTimeInMovie` (page 1342), `MakeTrackTimeTable` (page 1382), or `MakeMediaTimeTable` (page 1380). Your derived media handler returns the time table for the media in this block. Your media handler has to resize the handle.

startTime

The first point of the media to be included in the time table. This time value is expressed in the media's time coordinate system.

endTime

The last point of the media to be included in the time table. This time value is expressed in the media's time coordinate system.

timeIncrement

The resolution of the time table. The values in a time table are for a points in the media, and these points are separated by the amount of time specified by this parameter. The time value is expressed in the media's time coordinate system.

firstDataRefIndex

The first in the range of data reference indexes you are querying.

lastDataRefIndex

The last in the range of data reference indexes you are querying.

retDataRefSkew

A pointer to the number of entries, i.e., the number of entries in the offset table per data reference.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The Movie Toolbox calls your derived media handler's `MediaMakeMediaTimeTable` function whenever an application or other software calls the Toolbox's `QTMovieNeedsTimeTable` (page 1450), `GetMaxLoadedTimeInMovie` (page 1342), `MakeTrackTimeTable` (page 1382), or `MakeMediaTimeTable` (page 1380) function. When an application or other software calls one of these functions, it allocates an unlocked relocatable memory block for the time table to be returned and passes a handle to it in the `offsets` parameter. Your derived media handler must resize the block to accommodate the time table it returns.

The time table your derived media handler returns is a two-dimensional array of long integers that is organized so that each row in the table contains values for one data reference. The first column in the table contains values for the time in the media specified by the `startTime` parameter, and each subsequent column contains values for the point in the media that is later by the value specified by the `timeIncrement` parameter. Each long integer value in the table specifies the offset, in bytes, from the beginning of the data reference for that point in the media.

Special Considerations

The number of columns in the table returned by your derived media handler must be equal to `(endTime - startTime) / timeIncrement`, rounded up. It must also return the offset to the next row of the time table, in long integers, in the `retdataRefSkew` parameter. Because of alignment issues, this value is not always equal to `(endTime - startTime) / timeIncrement` rounded up.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MediaHandlers.h`

MediaMCIsPlayerEvent

Undocumented

```
ComponentResult MediaMCIsPlayerEvent (
   MediaHandler mh,
   const EventRecord *e,
   Boolean *handledIt
);
```

Parameters

mh
A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

e
A pointer to an `EventRecord` structure.

handledIt
A pointer to a Boolean; it is TRUE if the event was handled, FALSE otherwise.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MediaHandlers.h`

MediaNavigateTargetRefCon

Locates the object for keyboard focus.

```
ComponentResult MediaNavigateTargetRefCon (
   MediaHandler mh,
   long navigation,
   long *refCon
);
```

Parameters

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

navigation

Flags (see below) that define the direction of navigation. These flags are set by the standard controller, which follows the user's interaction with the tab key, shift key, and mouse. See these constants:

`kRefConNavigationNext`

`kRefConNavigationPrevious`

refCon

Returns a reference constant representing an object you're interested in. If this reference constant is not 0, your media handler will receive calls to `MediaRefConSetProperty` (page 1136) and `MediaRefConGetProperty` (page 1135).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 6.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`MediaHandlers.h`

MediaPrePrerollBegin

Undocumented

```
ComponentResult MediaPrePrerollBegin (
   MediaHandler mh,
   TimeValue time,
   Fixed rate,
   PrePrerollCompleteUPP completeProc,
   void *refcon
);
```

Parameters

mh

A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

time

Undocumented

rate

Undocumented

completeProc

A `PrePrerollCompleteProc` callback.

refcon

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

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MediaHandlers.h`

MediaPrePrerollCancel

Cancels a media handler pre-preroll operation that was started by MediaPrePrerollBegin.

```
ComponentResult MediaPrePrerollCancel (
   MediaHandler mh,
   void *refcon
);
```

Parameters

mh

A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

refcon

The reference constant that was passed to your `PrePrerollCompleteProc` callback.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MediaHandlers.h`

MediaPreroll

Prepares a media handler for playback.

```
ComponentResult MediaPreroll (
   MediaHandler mh,
   TimeValue time,
   Fixed rate
);
```

Parameters

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

time

The starting time of the media segment to play. This time value is expressed in your movie's time scale.

rate

The rate at which the Movie Toolbox expects to play the media. This is a 32-bit, fixed-point number. Positive values indicate forward rates; negative values correspond to reverse rates.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MediaHandlers.h`

MediaPutMediaInfo

Lets a derived media handler store proprietary information in its media.

```
ComponentResult MediaPutMediaInfo (
   MediaHandler mh,
   Handle h
);
```

Parameters

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

h

A handle to storage into which your media handler may place its proprietary information. You determine the format and content of the data that you store in this handle. Your media handler must resize the handle as appropriate before you exit this function. Do not dispose of this handle; it is owned by the Movie Toolbox. The Movie Toolbox uses the base media handler to write this data to your media.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Whenever the Movie Toolbox opens your media, it provides this private data to your media handler by calling your `MediaGetMediaInfo` (page 1105) function. Note that the Movie Toolbox may call this function before it calls your `MediaInitialize` (page 1127) function. Your derived media handler should support this function if you need to store private data in your media.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MediaHandlers.h`

MediaQueueNonPrimarySourceData

Undocumented

```
ComponentResult MediaQueueNonPrimarySourceData (
   MediaHandler mh,
   long inputIndex,
   long dataDescriptionSeed,
   Handle dataDescription,
   void *data,
   long dataSize,
   ICMCompletionProcRecordPtr asyncCompletionProc,
   const ICMFrameTimeRecord *frameTime,
   ICMConvertDataFormatUPP transferProc,
   void *refCon
);
```

Parameters

mh

A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

inputIndex

The ID of the entry in the media's input map to which the queued data corresponds.

dataDescriptionSeed

Undocumented

dataDescription

Undocumented

data

Undocumented

dataSize

Undocumented

asyncCompletionProc

A pointer to an `ICMCompletionProcRecord` structure.

frameTime

A pointer to an `ICMFrameTimeRecord` structure.

`transferProc`

Undocumented

`refCon`

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MediaHandlers.h`

MediaRefConGetProperty

Returns the current media handler state based on the property type.

```
ComponentResult MediaRefConGetProperty (
   MediaHandler mh,
   long refCon,
   long propertyType,
   void *propertyValue
);
```

Parameters

`mh`

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

`refCon`

The reference constant set by the media handler in `MediaNavigateTargetRefCon` (page 1131).

`propertyType`

The property type sent from the standard controller. Property type values are listed below. See these constants:

`kRefConPropertyCanHaveFocus`

`kRefConPropertyHasFocus`

`propertyValue`

A pointer to the value that was assigned. Its size is based on the `property` type.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This routine is called with the reference constant set in `MediaNavigateTargetRefCon` (page 1131) to get the current media handler state for a given property type.

Version Notes

Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`MediaHandlers.h`

MediaRefConSetProperty

Sets a new media handler state based on the property type.

```
ComponentResult MediaRefConSetProperty (
   MediaHandler mh,
   long refCon,
   long propertyType,
   void *propertyValue
);
```

Parameters

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

refCon

The reference constant set by the media handler in `MediaNavigateTargetRefCon` (page 1131).

propertyType

The property type sent from the standard controller. Property type values are listed below. See these constants:

`kRefConPropertyCanHaveFocus`

`kRefConPropertyHasFocus`

propertyValue

A pointer to the value to assign. Its size is based on the `property` type.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function is called with the reference constant that was set by `MediaNavigateTargetRefCon` (page 1131) to set a new media handler state for a given property type.

Version Notes
Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`MediaHandlers.h`

MediaReleaseSampleDataPointer

Balances calls to MediaGetSampleDataPointer to release allocated memory.

```
ComponentResult MediaReleaseSampleDataPointer (
   MediaHandler mh,
   long sampleNum
);
```

Parameters

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

sampleNum

The number of the sample that is to be released.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function should be used only by derived media handlers.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MediaHandlers.h`

MediaResolveTargetRefCon

Undocumented

```
ComponentResult MediaResolveTargetRefCon (
   MediaHandler mh,
   QTAtomContainer container,
   QTAtom atom,
   long *targetRefCon
);
```

Parameters

mh

A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

container

Undocumented

atom

Undocumented

targetRefCon

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MediaHandlers.h`

MediaSampleDescriptionB2N

Undocumented

```
ComponentResult MediaSampleDescriptionB2N (
   MediaHandler mh,
   SampleDescriptionHandle sampleDescriptionH
);
```

Parameters

`mh`
: A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

`sampleDescriptionH`
: A handle to a `SampleDescription` structure.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MediaHandlers.h`

MediaSampleDescriptionChanged

Informs a media handler that SetMediaSampleDescription has been called for a specified sample description.

```
ComponentResult MediaSampleDescriptionChanged (
   MediaHandler mh,
   long index
);
```

Parameters

`mh`
: The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

`index`
: The index of the sample description that has been changed.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MediaHandlers.h`

MediaSampleDescriptionN2B

Undocumented

```
ComponentResult MediaSampleDescriptionN2B (
   MediaHandler mh,
   SampleDescriptionHandle sampleDescriptionH
);
```

Parameters

mh
: A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

sampleDescriptionH
: *Undocumented*

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MediaHandlers.h`

MediaSetActionsCallback

Sets an ActionsProc callback for a media handler.

```
ComponentResult MediaSetActionsCallback (
   MediaHandler mh,
   ActionsUPP actionsCallbackProc,
   void *refcon
);
```

Parameters

mh
: A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

actionsCallbackProc
: A Universal Procedure Pointer to an `ActionsProc` callback.

refcon
: A pointer to a reference constant to be passed to your callback. Use this constant to point to a data structure containing any information your function needs.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MediaHandlers.h`

MediaSetActive

Enables and disables media.

```
ComponentResult MediaSetActive (
   MediaHandler mh,
   Boolean enableMedia
);
```

Parameters

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

enableMedia

A Boolean value that indicates whether your media is enabled or disabled. If this parameter is set to TRUE, your media is enabled; if the parameter is FALSE, your media is disabled.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Your derived media handler should support this function if you perform your own scheduling or if your media handler uses significant amounts of temporary storage. If you are doing your own scheduling (that is, you have set the `handlerNoScheduler` flag to 1 in the `flags` parameter of the `MediaSetHandlerCapabilities` (page 1145) function), your media handler needs to keep account of the media's active state so that you can properly respond to Movie Toolbox requests. When your media is disabled, you may choose to dispose of temporary storage you have allocated, so that the storage is available to other programs.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
SurfaceVertexProgram

Declared In
`MediaHandlers.h`

MediaSetChunkManagementFlags

Sets application-global flags that control media chunk management.

```
ComponentResult MediaSetChunkManagementFlags (
   MediaHandler mh,
   UInt32 flags,
   UInt32 flagsMask
);
```

Parameters

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

flags

Constants (see below) that determine chunk management. See these constants:

`kEmptyPurgableChunksOverAllowance`

flagsMask

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 6. Can be used only with Mac OS X 10.1 and later.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`MediaHandlers.h`

MediaSetClip

Specifies changes to a derived media handler's clipping region.

```
ComponentResult MediaSetClip (
   MediaHandler mh,
   RgnHandle theClip
);
```

Parameters

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

theClip

A handle to your media's clipping region. Your media handler is responsible for disposing of this region when you are done with it. Note that this region is defined in the movie's coordinate system.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Your derived media handler should support this function if you draw during playback. The Movie Toolbox calls this function only if you have set the `handlerHasSpatial` and `handlerCanClip` flags to 1 in the `flags` parameter of the `MediaSetHandlerCapabilities` (page 1145) function.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MediaHandlers.h`

MediaSetDimensions

Informs a media handler when its media's spatial dimensions change.

```
ComponentResult MediaSetDimensions (
   MediaHandler mh,
   Fixed width,
   Fixed height
);
```

Parameters

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

width

The width, in pixels, of the track rectangle. This field, along with the `height` field, specifies a rectangle that surrounds the image that is displayed when the current media is played. This value corresponds to the x coordinate of the lower-right corner of the rectangle and is expressed as a fixed-point number.

height

The height, in pixels, of the track rectangle. This value corresponds to the y coordinate of the lower-right corner of the rectangle and is expressed as a fixed-point number.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
You obtain the initial dimension information from the `width` and height fields of the `GetMovieCompleteParams` structure that the Movie Toolbox provides to your `MediaInitialize` (page 1127) function. Your derived media handler should support this function if you draw during playback.

Special Considerations
The Movie Toolbox calls this function only if you have set the `handlerHasSpatial` flag to 1 in the `flags` parameter of the `MediaSetHandlerCapabilities` (page 1145) function.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MediaHandlers.h`

MediaSetDoMCActionCallback

Sets a DoMCActionProc callback for a media handler.

```
ComponentResult MediaSetDoMCActionCallback (
   MediaHandler mh,
   DoMCActionUPP doMCActionCallbackProc,
   void *refcon
);
```

Parameters

mh
A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

doMCActionCallbackProc
A Universal Procedure Pointer to a `DoMCActionProc` callback.

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

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MediaHandlers.h`

MediaSetGraphicsMode

Sets the graphics mode and blend color of any media handler.

```
ComponentResult MediaSetGraphicsMode (
   MediaHandler mh,
   long mode,
   const RGBColor *opColor
);
```

Parameters

mh
The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

mode
The graphics mode of the media handler; see `Graphics Transfer Modes`.

opColor

A pointer to the color for use in blending and transparent operations. The media handler passes this color to QuickDraw as appropriate when you draw in `addPin`, `subPin`, `blend`, `transparent`, or `graphicsModeStraightAlphaBlend` mode.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qtactiontargets

qtactiontargets.win

vrmovies

vrmovies.win

vrscript.win

Declared In

`MediaHandlers.h`

MediaSetGWorld

Lets a derived media handler learn about changes to its media's graphic environment.

```
ComponentResult MediaSetGWorld (
   MediaHandler mh,
   CGrafPtr aPort,
   GDHandle aGD
);
```

Parameters

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

aPort

A pointer to the new graphics port. Note that this may be either a color or a black-and-white port.

aGD

A handle to the new graphics device.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Your derived media handler should support this function if you perform specialized graphics processing or if you are using the Image Compression Manager to decompress your media. Note that when the Movie Toolbox calls your `MediaIdle` (page 1125) function, it supplies you with information about the current graphics environment. Consequently, you do not need to support the `MediaSetGWorld` function in order to draw during playback. However, if your media data is compressed and you are using the Image Compression Manager to decompress sequences, you may need to provide updated graphics environment information before playback.

You obtain the initial graphics environment information from the `moviePort` and `movieGD` fields of the `GetMovieCompleteParams` structure that the Movie Toolbox provides to your `MediaInitialize` (page 1127) function. The Movie Toolbox calls this function only if you have set the `handlerHasSpatial` flag to 1 in the `flags` parameter of the `MediaSetHandlerCapabilities` (page 1145) function.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MediaHandlers.h`

MediaSetHandlerCapabilities

Lets a derived media handler report its capabilities to the base media handler.

```
ComponentResult MediaSetHandlerCapabilities (
   MediaHandler mh,
   long flags,
   long flagsMask
);
```

Parameters

mh
A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

flags
Flags (see below) that specify the capabilities of your derived media handler. This parameter contains a number of flags, each of which corresponds to a particular feature. You may work with more than one flag at a time. Be sure to set unused flags to 0. See these constants:

`handlerHasSpatial`

`handlerCanClip`

`handlerCanMatte`

`handlerCanTransferMode`

`handlerNeedsBuffer`

`handlerNoIdle`

`handlerNoScheduler`

`handlerWantsTime`

`handlerCGrafPortOnly`

flagsMask
Indicates which flags in the `flags` parameter are to be considered in this operation. For each bit in the `flags` parameter that you want the base media handler to consider, you must set the corresponding bit in the `flagsMask` parameter to 1. Set unused flags to 0. This allows you to work with a single flag without altering the settings of other flags.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Your media handler may call this function at any time. In general, you should call it from your `MediaInitialize` (page 1127) function , so that you report your capabilities to the base media handler before the Movie Toolbox starts working with your media. You may call this function again later, in response to changing conditions. For example, if your media handler receives a matrix that it cannot accommodate from the `MediaSetMatrix` (page 1147) function, you can allow the base media handler to handle your drawing by calling this function and setting the `handlerNeedsBuffer` flag in both the `flags` parameter and the `flagsMask` parameter to 1.

Special Considerations

Note that this function is provided by the base media handler; your media handler does not support this function.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
AlwaysPreview

qdmediahandler

qdmediahandler.win

Declared In
`MediaHandlers.h`

MediaSetHints

Implements the appropriate behavior for the various media hints such as scrub mode and high-quality mode.

```
ComponentResult MediaSetHints (
   MediaHandler mh,
   long hints
);
```

Parameters

`mh`

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

`hints`

All hint bits that currently apply to the given media.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
When an application calls `SetMoviePlayHints` (page 1485) or `SetMediaPlayHints` (page 1475), your media handler's `MediaSetHints` routine is called.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MediaHandlers.h`

MediaSetMatrix

Tells a media handler about changes to either the movie matrix or the track matrix.

```
ComponentResult MediaSetMatrix (
   MediaHandler mh,
   MatrixRecord *trackMovieMatrix
);
```

Parameters

mh
: The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

trackMovieMatrix
: A pointer to the matrix that transforms your media's pixels into the movie's coordinate system. The Movie Toolbox obtains this matrix by concatenating the track matrix and the movie matrix. You should use this matrix whenever you are displaying graphical data from your media.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
You obtain the initial matrix from the `trackMovieMatrix` field of the `GetMovieCompleteParams` structure that the Movie Toolbox provides to your `MediaInitialize` (page 1127) function. Your derived media handler should support this function if you draw during playback.

Special Considerations
The Movie Toolbox calls this function only if you have set the `handlerHasSpatial` flag to 1 in the `flags` parameter of the `MediaSetHandlerCapabilities` (page 1145) function.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MediaHandlers.h`

MediaSetMediaTimeScale

Informs a media handler that its media's time scale has been changed.

```
ComponentResult MediaSetMediaTimeScale (
   MediaHandler mh,
   TimeScale newTimeScale
);
```

Parameters

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

newTimeScale

Specifies your media's new time scale.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

You obtain the initial media time scale information from the `mediaScale` field of the `GetMovieCompleteParams` structure that the Movie Toolbox provides to your `MediaInitialize` (page 1127) function.

Special Considerations

Your derived media handler should support this function if your media handler stores time information that pertains to its media.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MediaHandlers.h`

MediaSetMovieTimeScale

Informs a media handler that the movie's time scale has been changed.

```
ComponentResult MediaSetMovieTimeScale (
   MediaHandler mh,
   TimeScale newTimeScale
);
```

Parameters

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

newTimeScale

The movie's new time scale.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion
You obtain the initial movie time scale information from the `movieScale` field of the `GetMovieCompleteParams` structure that the Movie Toolbox provides to your `MediaInitialize` (page 1127) function.

Special Considerations

Your derived media handler should support this function if your media handler stores time information in the movie's time coordinate system.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MediaHandlers.h`

MediaSetNonPrimarySourceData

Allows a media handler to support receiving media data from other media handlers.

```
ComponentResult MediaSetNonPrimarySourceData (
   MediaHandler mh,
   long inputIndex,
   long dataDescriptionSeed,
   Handle dataDescription,
   void *data,
   long dataSize,
   ICMCompletionProcRecordPtr asyncCompletionProc,
   ICMConvertDataFormatUPP transferProc,
   void *refCon
);
```

Parameters

mh
: The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

inputIndex
: This value is the ID of the entry in the media's input map to which the data provided by the call corresponds.

dataDescriptionSeed
: This value is changed each time the `dataDescription` has changed. This allows for a quick check by the media handler to see if the `dataDescription` has changed.

dataDescription
: A handle to a data structure describing the input data.

data
: A pointer to the input data. This pointer must contain a 32-bit address.

dataSize
: The size of the sample in bytes.

asyncCompletionProc

A pointer to a `ICMCompletionProcRecord` structure. If `asyncCompletionProc` is set to `NIL`, the data pointer will be valid only for the duration of this call. If `asyncCompletionProc` is not `NIL`, it contains an `ICMCompletionProcRecord` structure that must be called when your media handler is done with the provided data pointer.

transferProc

A routine that allows the application to transform the type of the input data to the kind of data preferred by the codec. The client of the codec passes the source data in the form most convenient for it. If the codec needs the data in another form, it can negotiate with the caller or directly with the Image Compression Manager to obtain the required data format.

refCon

A reference constant, defined as a void pointer. Your application specifies the value of this reference constant in the function structure you pass to the media handler.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

There are two tasks required to support modifier tracks in a derived media handler: sending and receiving. The base media handler takes care of sending data for all its clients. Therefore, authors of derived media handlers do not usually need to implement sending data support.

Receiving data is a more complex situation. The base media handler takes care of input types that it understands. The base media handler supports the following types of data: If a media handler wants to support receiving other types of data it must implement `MediaSetNonPrimarySourceData`. `MediaSetNonPrimarySourceData` is called by modified tracks to supply the current data for each input. All unrecognized input types should be delegated to the base media handler so that they can be handled.

The following is a basic shell implementation of a derived media handler's `MediaSetNonPrimarySourceData` function. Note that your derived media handler must delegate all input types it does not handle to the base media handler:

```
// MySetNonPrimarySourceData coding example
kTrackModifierTypeMatrix
kTrackModifierTypeGraphicsMode
kTrackModifierTypeClip
kTrackModifierTypeVolume
kTrackModifierTypeBalance
pascal ComponentResult MySetNonPrimarySourceData( MyGlobals store,
    long inputIndex, long dataDescriptionSeed, Handle dataDescription,
    void *data, long dataSize,
    ICMCompletionProcRecordPtr asyncCompletionProc,
    UniversalProcPtr transferProc, void *refCon )
{
ComponentResult err =noErr;
QTAtom inputAtom;
QTAtom typesAtom;
long inputType;
// determine what kind of input this is
inputAtom =QTFindChildByID(store->
inputMap,
    kParentAtomIsContainer, kTrackModifierInput, inputIndex, NIL);
if (!inputAtom) {
    err =cannotFindAtomErr;
    goto bail;
}
```

```
    typesAtom =QTFindChildByID(store->
inputMap, inputAtom,
    kTrackModifierType, 1, NIL);
err =QTCopyAtomDataToPtr(store->
inputMap, typesAtom, FALSE,
    sizeof(inputType), &inputType, NIL);
if (err) goto bail;
switch(inputType) {
    case kMyInputType:
        if (data ==NIL) {
            // no data, reset to default value
        }
        else {
            // use this data
            // when done, notify caller we're done with this data
            if (asyncCompletionProc)
                CallICMCompletionProc(
                    asyncCompletionProc->
completionProc,
                    noErr, codecCompletionSource | codecCompletionDest,
                    asyncCompletionProc->
completionRefCon);
        }
        break;
```

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MediaHandlers.h`

MediaSetPublicInfo

Undocumented

```
ComponentResult MediaSetPublicInfo (
   MediaHandler mh,
   OSType infoSelector,
   void *infoDataPtr,
   Size dataSize
);
```

Parameters

mh
: The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

infoSelector
: *Undocumented*

infoDataPtr
: *Undocumented*

dataSize

Undocumented

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qtskins

qtskins.win

Declared In
`MediaHandlers.h`

MediaSetPurgeableChunkMemoryAllowance

Sets the maximum amount of memory that QuickTime will allow purgeable chunks to occupy.

```
ComponentResult MediaSetPurgeableChunkMemoryAllowance (
   MediaHandler mh,
   Size allowance
);
```

Parameters

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

allowance

The number of bytes allowed.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This is an application-global setting.

Version Notes
Introduced in QuickTime 6. Can be used only with Mac OS X 10.1 and later.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`MediaHandlers.h`

MediaSetRate

Sets a media's playback rate.

```
ComponentResult MediaSetRate (
   MediaHandler mh,
   Fixed rate
);
```

Parameters

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

rate

A 32-bit, fixed-point number that indicates your media's new effective playback rate. This effective rate accounts for any master time bases that may be in use with the current movie. Positive values represent forward rates and negative values indicate reverse rates.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

You obtain the initial rate information from the `effectiveRate` field of the `GetMovieCompleteParams` structure that the Movie Toolbox provides to your `MediaInitialize` (page 1127) function. Your derived media handler should support this function if you perform your own scheduling; that is, you have set the `handlerNoScheduler` flag to 1 in the `flags` parameter of `MediaSetHandlerCapabilities` (page 1145). Your media handler can use this function to determine when your media is playing, and the direction and rate of playback. This information can help you prepare for playback more efficiently.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MediaHandlers.h`

MediaSetScreenLock

Locks the display screen for a media handler.

```
ComponentResult MediaSetScreenLock (
   MediaHandler mh,
   Boolean lockIt
);
```

Parameters

mh

A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

lockIt

Pass TRUE to lock the screen, FALSE to unlock it.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MediaHandlers.h`

MediaSetSoundBalance

Sets the right/left sound balance of a track.

```
ComponentResult MediaSetSoundBalance (
   MediaHandler mh,
   short balance
);
```

Parameters

mh
A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

balance
The new balance setting for the media handler's track. The balance setting is a signed 16-bit integer that controls the relative volume of the left and right sound channels. A value of 0 sets the balance to neutral. Positive values shift the balance to the right channel, negative values to the left channel. The valid range is 127 (right channel only) to -128 (left channel only).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Use this function to alter the audio balance on a per-track basis. It works with both mono and stereo sources. The actual volume of the audio is not changed by this setting, only its relative distribution. This function operates on sound tracks, music tracks, and Flash tracks containing audio. It may operate on other audio track types as well; if the the chosen media handler does not support this function, it returns `badComponentSelector`. To obtain the media handler for a track, call `GetTrackMedia` (page 1612) and then `GetMediaHandler` (page 1577).

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
vrscript
vrscript.win

Declared In
`MediaHandlers.h`

MediaSetSoundBassAndTreble

Sets the bass and treble controls for a media handler.

```
ComponentResult MediaSetSoundBassAndTreble (
   MediaHandler mh,
   short bass,
   short treble
);
```

Parameters

mh

A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

bass

Undocumented

treble

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MediaHandlers.h`

MediaSetSoundEqualizerBands

Sets sound equalizer bands for a media handler.

```
ComponentResult MediaSetSoundEqualizerBands (
   MediaHandler mh,
   MediaEQSpectrumBandsRecordPtr spectrumInfo
);
```

Parameters

mh

A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

spectrumInfo

A pointer to a `MediaEQSpectrumBandsRecord` structure.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

sndequalizer

SurfaceVertexProgram

Declared In
`MediaHandlers.h`

MediaSetSoundLevelMeteringEnabled

Enables or disables sound level metering for a media handler.

```
ComponentResult MediaSetSoundLevelMeteringEnabled (
   MediaHandler mh,
   Boolean enable
);
```

Parameters

mh
: A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

enable
: Pass TRUE to enable sound level metering, FALSE to disable it.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
SurfaceVertexProgram

Declared In
`MediaHandlers.h`

MediaSetSoundLocalizationData

Supports 3D sound capabilities in a media handler that plays sound.

```
ComponentResult MediaSetSoundLocalizationData (
   MediaHandler mh,
   Handle data
);
```

Parameters

mh
: The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

data
: The data passed to your media handler, in the format of a Macintosh Sound Sprockets `SSpLocalizationData` structure.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This routine is passed a handle containing the new `SSpLocalizationData` structure to use. If the handle is `NIL`, it indicates that no 3D sound effects should be used. If you implement this routine, and return `noErr` as the result, it is assumed that your media handle assumes responsibility for disposing of the data handle passed. If the implementation of this routine returns an error, the caller will dispose of the handle. The reason for this behavior is to minimize the copying of the settings handle, making it easier for developers to implement this function. This function is called regardless of whether the 3D sound settings were set on the track using `SetTrackSoundLocalizationSettings` (page 1655) or via a modifier track mechanism.

Special Considerations

If you are creating a media handler that plays sound and wish to support 3D sound capabilities, you need to implement this routine.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MediaHandlers.h`

MediaSetSoundOutputComponent

Sets the sound output component for a media handler.

```
ComponentResult MediaSetSoundOutputComponent (
   MediaHandler mh,
   Component outputComponent
);
```

Parameters

mh
A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

outputComponent
An instance of a sound output component. Your software obtains this reference when calling `OpenComponent` or `OpenDefaultComponent`.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MediaHandlers.h`

MediaSetTrackInputMapReference

Provides a derived media handler with an updated input map.

```
ComponentResult MediaSetTrackInputMapReference (
   MediaHandler mh,
   QTAtomContainer inputMap
);
```

Parameters

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

inputMap

The media input map for this operation. Do not modify or dispose of the input map provided.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

When an application modifies the media input map, this function provides the derived media handler with the updated input map. When this function is called, the media handler should store the updated input map and recheck the types of all inputs, if it is caching this information. The input map reference passed to this function should not be disposed of or modified by the media handler.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MediaHandlers.h`

MediaSetUserPreferredCodecs

Requests that a media handler favor specified codec components when selecting components with which to play media.

```
ComponentResult MediaSetUserPreferredCodecs (
   MediaHandler mh,
   CodecComponentHandle userPreferredCodecs
);
```

Parameters

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

userPreferredCodecs

A handle containing component identifiers. The media handler component will make its own copy of this handle. The components should be specified in order from most preferred to least preferred. Pass `NIL` to invalidate the standing request without substituting another.

Return Value

See `Error Codes`. Returns `badComponentSelector` if the media handler component does not support this call. Returns `noErr` if there is no error.

Discussion

This method does not guarantee that the specified components will be used; other factors may take precedence. Components that are preferred may not be used if they can't be part of the chain required to play the media; for example, if they don't handle the pixel format or the video output.

Here is an example of code that uses this function:

```
CodecComponentHandle userPreferredCodecs =nil;
ComponentDescription cd ={ decompressorComponentType,
                           myPreferredCodecType,
                           myPreferredCodecManufacturer,
                           0,
                           0 };
CodecComponent    c =FindNextComponent( 0, &cd );
MediaHandler      myMedia;
OSErr             err;
PtrToHand( &c, (Handle*)&userPreferredCodecs, sizeof(c) );
myMedia =GetMediaHandler( GetTrackMedia( track ) );
err =MediaSetUserPreferredCodecs( myMedia, userPreferredCodecs );
DisposeHandle( (Handle)userPreferredCodecs );
```

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MediaHandlers.h`

MediaSetVideoParam

Lets you dynamically adjust the brightness, contrast, hue, sharpness, saturation, black level, and white level of a video image.

```
ComponentResult MediaSetVideoParam (
   MediaHandler mh,
   long whichParam,
   unsigned short *value
);
```

Parameters

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

whichParam

A constant (see below) that specifies the video parameter you want to adjust. See these constants:

`kMediaVideoParamBrightness`

`kMediaVideoParamContrast`

`kMediaVideoParamHue`

`kMediaVideoParamSharpness`

`kMediaVideoParamSaturation`

`kMediaVideoParamBlackLevel`

`kMediaVideoParamWhiteLevel`

value

The actual value of the video parameter. The meaning of the values vary depending on the implementation.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MediaHandlers.h`

MediaTargetRefConsEqual

Undocumented

```
ComponentResult MediaTargetRefConsEqual (
   MediaHandler mh,
   long firstRefCon,
   long secondRefCon,
   Boolean *equal
);
```

Parameters

mh

A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

firstRefCon

Undocumented

secondRefCon

Undocumented

equal

A pointer to a Boolean; it is TRUE if `firstRefCon` and `secondRefCon` are equal, FALSE otherwise.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MediaHandlers.h`

MediaTimeBaseChanged

Undocumented

```
ComponentResult MediaTimeBaseChanged (
   MediaHandler mh
);
```

Parameters

mh

A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

Return Value
Undocumented

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MediaHandlers.h`

MediaTrackEdited

Informs a derived media handler about edits to its track.

```
ComponentResult MediaTrackEdited (
   MediaHandler mh
);
```

Parameters

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Your derived media handler should support this function if you are caching location information about track edits, or if you are using any time values in the movie's time base. Whenever the Movie Toolbox calls this function, your media handler should recalculate this type of information.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MediaHandlers.h`

MediaTrackPropertyAtomChanged

Notifies the derived media handler whenever its media property atom has changed.

```
ComponentResult MediaTrackPropertyAtomChanged (
   MediaHandler mh
);
```

Parameters

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
`MediaTrackPropertyAtomChanged` is called whenever `SetMediaPropertyAtom` (page 1476) is called. If the media handler uses information from the property atom, it should rebuild the information at this time.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MediaHandlers.h`

MediaTrackReferencesChanged

Notifies the derived media handler whenever the track references in the movie change.

```
ComponentResult MediaTrackReferencesChanged (
   MediaHandler mh
);
```

Parameters

mh

The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
When an application creates, modifies, or deletes a track reference, the media handler's `MediaTrackReferencesChanged` function is called. When this function is called, a media handler should rebuild all information about track references and reset its values for all media inputs to their default values.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MediaHandlers.h`

MediaVideoOutputChanged

Undocumented

```
ComponentResult MediaVideoOutputChanged (
   MediaHandler mh,
   ComponentInstance vout
);
```

Parameters

`mh`
The Toolbox's connection to your derived media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

`vout`
Undocumented

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MediaHandlers.h`

NewPrePrerollCompleteUPP

Allocates a Universal Procedure Pointer for the PrePrerollCompleteProc callback.

```
PrePrerollCompleteUPP NewPrePrerollCompleteUPP (
   PrePrerollCompleteProcPtr userRoutine
);
```

Parameters

`userRoutine`
A pointer to your application-defined function.

Return Value
A new UPP; see `Universal Procedure Pointers`.

Discussion
This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes
Introduced in QuickTime 4.1. Replaces `NewPrePrerollCompleteProc`.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`MediaHandlers.h`

Callbacks

PrePrerollCompleteProc

Undocumented

```
typedef void (*PrePrerollCompleteProcPtr) (MediaHandler mh, OSErr err, void *refcon);
```

If you name your function `MyPrePrerollCompleteProc`, you would declare it this way:

```
void MyPrePrerollCompleteProc (
    MediaHandler    mh,
    OSErr           err,
    void            *refcon );
```

Parameters

mh
A media handler.

err
An error code; see `Error Codes`.

refcon
Pointer to a reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

Declared In
`MediaHandlers.h`

Data Types

GetMovieCompleteParams

Defines the layout of the complete movie parameter structure used by MediaInitialize.

```
struct GetMovieCompleteParams {
     short              version;
     Movie              theMovie;
     Track              theTrack;
     Media              theMedia;
     TimeScale          movieScale;
     TimeScale          mediaScale;
     TimeValue          movieDuration;
     TimeValue          trackDuration;
     TimeValue          mediaDuration;
     Fixed              effectiveRate;
     TimeBase           timeBase;
     short              volume;
     Fixed              width;
     Fixed              height;
     MatrixRecord       trackMovieMatrix;
     CGrafPtr           moviePort;
     GDHandle           movieGD;
     PixMapHandle       trackMatte;
     QTAtomContainer    inputMap;
 };
```

Fields

`version`

Discussion

Specifies the version of this structure. This field is always set to 1.

`theMovie`

Discussion

Identifies the movie that contains the current media's track. This movie identifier is supplied by the Movie Toolbox. Your component may use this identifier to obtain information about the movie that is using your media.

`theTrack`

Discussion

Identifies the track that contains the current media. This track identifier is supplied by the Movie Toolbox. Your component may use this identifier to obtain information about the track that contains your media. For example, you might call `GetTrackNextInterestingTime` (page 1373) to examine the track's edit list.

`theMedia`

Discussion

Identifies the current media. This media identifier is supplied by the Movie Toolbox. Your derived media handler can use this identifier to read samples or sample descriptions from the current media, using `GetMediaSample` (page 1583) and `GetMediaSampleDescription` (page 1587).

`movieScale`

Discussion

Specifies the time scale of the movie that contains the current media's track. If the Movie Toolbox changes the movie's time scale, the toolbox calls your derived media handler's `MediaSetMovieTimeScale` (page 1148) function.

`mediaScale`

Discussion

Specifies the time scale of the current media. If the Movie Toolbox changes your media's time scale, the toolbox calls your derived media handler's `MediaSetMediaTimeScale` (page 1147) function.

`movieDuration`

Discussion
Contains the movie's duration. This value is expressed in the movie's time scale.

`trackDuration`

Discussion
Contains the track's duration. This value is expressed in the movie's time scale.

`mediaDuration`

Discussion
Contains the media's duration. This value is expressed in the media's time scale.

`effectiveRate`

Discussion
Contains the media's effective rate. This rate ties the media's time scale to the passage of absolute time, and does not necessarily correspond to the movie's rate. This value takes into account any master time bases that may be serving the media's time base. The value of this field indicates the number of time units (in the media's time scale) that pass each second. This rate is represented as a 32-bit, fixed-point number. The high-order 16 bits contain the integer portion, and the low-order 16 bits contain the fractional portion. The rate is negative when time is moving backward for the media. Whenever the Movie Toolbox changes your media's effective rate, it calls your derived media handler's `MediaSetRate` (page 1152) function.

`timeBase`

Discussion
Identifies the media's time base.

`volume`

Discussion
Contains the media's current volume setting. This value is represented as a 16-bit, fixed-point number. The high-order 8 bits contain the integer portion; the low-order 8 bits contain the fractional part. Volume values range from -1.0 to 1.0. Negative values play no sound but preserve the absolute value of the volume setting. If QuickTime changes your media's volume, it calls your derived media handler's `MediaGSetVolume` (page 1122) function.

`width`

Discussion
Indicates the width, in pixels, of the track rectangle. This field, along with the `height` field, specifies a rectangle that surrounds the image that is displayed when the current media is played. This value corresponds to the x coordinate of the lower-right corner of the rectangle and is expressed as a fixed-point number. If the Movie Toolbox modifies this rectangle, the toolbox calls your derived media handler's `MediaSetDimensions` (page 1142) function. Note that your media need not present only a rectangular image. The Movie Toolbox can use a clipping region to cause your media's image to be displayed in a region of arbitrary shape, and it can use a matte to control the image's transparency. The toolbox calls your derived media handler's `MediaSetClip` (page 1141) function whenever it changes your media's clipping region. The `trackMatte` field in this structure specifies a matte region.

`height`

Discussion
Indicates the height, in pixels, of the track rectangle. This value corresponds to the y coordinate of the lower-right corner of the rectangle and is expressed as a fixed-point number.

`trackMovieMatrix`

Discussion
Specifies the matrix that transforms your media's pixels into the movie's coordinate system. The Movie Toolbox obtains this matrix by concatenating the track matrix and the movie matrix. You should use this matrix whenever you are displaying graphical data from your media. Whenever the Movie Toolbox modifies this matrix, it calls your derived media handler's `MediaSetMatrix` (page 1147) function.

`moviePort`

Discussion
Indicates the movie's graphics port. Whenever the Movie Toolbox changes the movie's graphics world, it calls your derived media handler's `MediaSetGWorld` (page 1144) function.

`movieGD`

Discussion
Specifies the movie's graphics device. Whenever the Movie Toolbox changes the movie's graphics world, it calls your derived media handler's `MediaSetGWorld` (page 1144) function.

`trackMatte`

Discussion
Identifies the matte region assigned to the track that uses your media. This field contains a handle to a pixel map that contains a blend matte. Your component is not responsible for disposing of this matte. If there is no matte, this field is set to `NIL`.

`inputMap`

Discussion
A reference to the media's input map. The media input map should not be modified or disposed.

Related Functions
`MediaInitialize` (page 1127)

Declared In
`MediaHandlers.h`

LevelMeterInfo

Contains sound level meter readings.

```
struct LevelMeterInfo {
     short    numChannels;
     UInt8    leftMeter;
     UInt8    rightMeter;
 };
```

Fields
`numChannels`

Discussion
Contains 1 for mono or 2 for stereo source.

`leftMeter`

Discussion
Left meter level, 0-255 range.

`rightMeter`

Discussion
Right meter level, 0-255 range.

Related Functions
`MediaGetSoundLevelMeterInfo` (page 1114)

Declared In
`MediaHandlers.h`

LevelMeterInfoPtr

Represents a type used by the Media Handler API.

```
typedef LevelMeterInfo * LevelMeterInfoPtr;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Sound.h`

MediaEQSpectrumBandsRecord

Provides data for MediaGetSoundEqualizerBands and MediaSetSoundEqualizerBands.

```
struct MediaEQSpectrumBandsRecord {
     short               count;
     UnsignedFixedPtr    frequency;
 };
```

Fields
`count`

Discussion
Number of frequencies in this structure.

`frequency`

Discussion
Pointer to array of frequencies.

Related Functions
`MediaGetSoundEqualizerBands` (page 1113)
`MediaSetSoundEqualizerBands` (page 1155)

Declared In
`MediaHandlers.h`

MediaEQSpectrumBandsRecordPtr

Represents a type used by the Media Handler API.

```
typedef MediaEQSpectrumBandsRecord * MediaEQSpectrumBandsRecordPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MediaHandlers.h`

PrePrerollCompleteUPP

Represents a type used by the Media Handler API.

```
typedef STACK_UPP_TYPE(PrePrerollCompleteProcPtr) PrePrerollCompleteUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MediaHandlers.h`

QTCustomActionTargetPtr

Represents a type used by the Media Handler API.

```
typedef QTCustomActionTargetRecord * QTCustomActionTargetPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MediaHandlers.h`

QTCustomActionTargetRecord

Defines a target for CallComponentExecuteWiredAction.

```
struct QTCustomActionTargetRecord {
     Movie            movie;
     DoMCActionUPP    doMCActionCallbackProc;
     long             callBackRefcon;
     Track            track;
     long             trackObjectRefCon;
     Track            defaultTrack;
     long             defaultObjectRefCon;
     long             reserved1;
     long             reserved2;
 };
```

Fields
`movie`

Discussion
A movie identifier obtained from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

`doMCActionCallbackProc`

Discussion
A Universal Procedure Pointer that accesses a `DoMCActionProc` callback.

`callBackRefcon`

Discussion
A reference constant to be passed to the `DoMCActionProc` callback. Use this value to point to a data structure containing any information the callback needs.

`track`

Discussion
A track identifier obtained from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

`trackObjectRefCon`

Discussion
Undocumented

`defaultTrack`

Discussion
The identifier of a default track, obtained from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

`defaultObjectRefCon`

Discussion
Undocumented

`reserved1`

Discussion
Reserved.

`reserved2`

Discussion
Reserved.

Declared In
`MediaHandlers.h`

Constants

MediaForceUpdate Values

Constants passed to MediaForceUpdate.

```
enum {
  forceUpdateRedraw              = 1 << 0,
  forceUpdateNewBuffer           = 1 << 1
};
```

Declared In
`MediaHandlers.h`

Data Handler Flags

Constants that represent data handler flags.

```
enum {
  handlerHasSpatial             = 1 << 0,
  handlerCanClip                = 1 << 1,
  handlerCanMatte               = 1 << 2,
  handlerCanTransferMode        = 1 << 3,
  handlerNeedsBuffer            = 1 << 4,
  handlerNoIdle                 = 1 << 5,
  handlerNoScheduler            = 1 << 6,
  handlerWantsTime              = 1 << 7,
  handlerCGrafPortOnly          = 1 << 8,
  handlerCanSend                = 1 << 9,
  handlerCanHandleComplexMatrix = 1 << 10,
  handlerWantsDestinationPixels = 1 << 11,
  handlerCanSendImageData       = 1 << 12,
  handlerCanPicSave             = 1 << 13
};
```

Declared In
`MediaHandlers.h`

MediaSetChunkManagementFlags Values

Constants passed to MediaSetChunkManagementFlags.

```
enum {
  kEmptyPurgableChunksOverAllowance = 1
};
```

Declared In
`MediaHandlers.h`

MediaSetVideoParam Values

Constants passed to MediaSetVideoParam.

```
enum {
  kMediaVideoParamBrightness    = 1,
  kMediaVideoParamContrast      = 2,
  kMediaVideoParamHue           = 3,
  kMediaVideoParamSharpness     = 4,
  kMediaVideoParamSaturation    = 5,
  kMediaVideoParamBlackLevel    = 6,
  kMediaVideoParamWhiteLevel    = 7
};
```

Declared In
`MediaHandlers.h`

MediaNavigateTargetRefCon Values

Constants passed to MediaNavigateTargetRefCon.

```
enum {
  kRefConNavigationNext          = 0,
  kRefConNavigationPrevious      = 1
};
```

Declared In
`MediaHandlers.h`

MediaRefConSetProperty Values

Constants passed to MediaRefConSetProperty.

```
enum {
  kRefConPropertyCanHaveFocus    = 1,    /* Boolean */
  kRefConPropertyHasFocus        = 2     /* Boolean */
};
```

Declared In
`MediaHandlers.h`

Media Task Flags

Constants that represent flags for media tasks.

```
enum {
  mDidDraw                       = 1 << 0,
  mNeedsToDraw                   = 1 << 2,
  mDrawAgain                     = 1 << 3,
  mPartialDraw                   = 1 << 4,
  mWantIdleActions               = 1 << 5
};
enum {
  mMustDraw                      = 1 << 3,
  mAtEnd                         = 1 << 4,
  mPreflightDraw                 = 1 << 5,
  mSyncDrawing                   = 1 << 6,
  mPrecompositeOnly              = 1 << 9,
  mSoundOnly                     = 1 << 10,
  mDoIdleActionsBeforeDraws      = 1 << 11,
  mDisableIdleActions            = 1 << 12
};
enum {
  mOpaque                        = 1L << 0,
  mInvisible                     = 1L << 1
};
```

Declared In
`MediaHandlers.h`

MediaHitTestTargetRefCon Values

Constants passed to MediaHitTestTargetRefCon.

```
enum {
  mHitTestBounds                = 1L << 0, /*    point must only be within
targetRefCon's bounding box */
  mHitTestImage                 = 1L << 1, /*  point must be within the shape of
the targetRefCon's image */
  mHitTestInvisible             = 1L << 2, /*  invisible targetRefCon's may be hit
 tested */
  mHitTestIsClick               = 1L << 3 /*  for codecs that want mouse events */
};
```

Declared In

`MediaHandlers.h`

Movie Controller Reference

Framework:	Frameworks/QuickTime.framework
Declared in	Movies.h, HIMovieView.h

Overview

Movie controllers provide a user interface for playing and editing movies, eliminating much of the complexity of working with movies. Movie controllers are implemented in QuickTime as components. This allows customized controllers to be plugged in to QuickTime for use by your application.

Functions by Task

Associating Movies With Controllers

`MCGetIndMovie` (page 1198)
: Lets your application to retrieve the movie reference for a movie that is associated with a movie controller.

`MCNewAttachedController` (page 1207)
: Associates a specified movie with a movie controller.

`MCSetMovie` (page 1218)
: Associates a movie with a specified movie controller.

Customizing Event Processing

`MCActivate` (page 1184)
: Lets a controller respond to activate, deactivate, suspend, and resume events.

`MCClick` (page 1187)
: Lets a controller respond when the user clicks in a movie controller window.

`MCDraw` (page 1190)
: Responds to an update event.

`MCIdle` (page 1202)
: Performs idle processing for a movie controller.

`MCKey` (page 1206)
: Handles keyboard events for a movie controller.

Editing Movies With a Controller

`MCClear` (page 1187)
: Removes the current movie selection from the movie associated with a specified controller.

`MCCopy` (page 1188)
: Returns a copy of the current movie selection from the movie associated with a specified controller.

`MCCut` (page 1189)
: Returns a copy of the current movie selection from the movie associated with a specified controller and then removes the current movie selection from the source movie.

`MCEnableEditing` (page 1192)
: Enables and disables editing for a movie controller.

`MCGetMenuString` (page 1199)
: Retrieves the text string for a movie controller menu command.

`MCIsEditingEnabled` (page 1204)
: Determines whether editing is currently enabled for a movie controller.

`MCPaste` (page 1208)
: Inserts a specified movie at the current movie time in the movie associated with a specified controller.

`MCSetUpEditMenu` (page 1219)
: Correctly highlights and names the items in your application's Edit menu.

`MCUndo` (page 1221)
: Lets your application discard the effects of the most recent edit operation.

Getting and Setting Movie Controller Time

`MCGetCurrentTime` (page 1196)
: Obtains the time value represented by the indicator on the movie controller's slider.

`MCSetDuration` (page 1217)
: Lets your application set a controller's duration in the case where a controller does not have a movie associated with it.

Handling Movie Events

`MCGetControllerInfo` (page 1195)
: Determines the current status of a movie controller and its associated movie, for menu highlighting.

`MCIsPlayerEvent` (page 1204)
: Handles all events for a movie controller.

`MCPtInController` (page 1209)
: Reports whether a point is in the control area of a movie.

Managing Controller Attributes

`MCDrawBadge` (page 1191)
: Displays a controller's badge.

`MCGetClip` (page 1193)
: Obtains information describing a movie controller's clipping regions.

`MCGetControllerBoundsRect` (page 1193)
: Returns a movie controller's boundary rectangle.

`MCGetControllerBoundsRgn` (page 1194)
: Returns the actual region occupied by the controller and its movie.

`MCGetControllerPort` (page 1196)
: Returns a movie controller's color graphics port.

`MCGetVisible` (page 1200)
: Returns a value that indicates whether or not a movie controller is visible.

`MCGetWindowRgn` (page 1201)
: Determines the window region that is actually in use by a controller and its movie.

`MCIsControllerAttached` (page 1203)
: Returns a value that indicates whether a movie controller is attached to its movie.

`MCPositionController` (page 1208)
: Controls the position of a movie and its controller on the computer display.

`MCSetClip` (page 1213)
: Lets you set a movie controller's clipping region.

`MCSetControllerAttached` (page 1214)
: Lets your application control whether a movie controller is attached to its movie or detached from it.

`MCSetControllerBoundsRect` (page 1215)
: Lets you change the position and size of a movie controller.

`MCSetControllerPort` (page 1216)
: Lets your application set the graphics port for a movie controller.

`MCSetVisible` (page 1220)
: Lets your application control the visibility of a movie controller.

Movie Controller Action Functions

`MCDoAction` (page 1190)
: Invokes a movie controller component and makes it perform a specified action.

`MCMovieChanged` (page 1206)
: Informs a movie controller component that your application has used the Movie Toolbox to change the characteristics of its associated movie.

`MCSetActionFilterWithRefCon` (page 1212)
: Establishes an action filter function for a movie controller.

Working With The Idle Manager

`MCSetIdleManager` (page 1218)
: Lets a movie controller component report its idling needs.

Supporting Functions

`DisposeMCActionFilterUPP` (page 1179)
    Disposes of a MCActionFilterUPP pointer.

`DisposeMCActionFilterWithRefConUPP` (page 1179)
    Disposes of a MCActionFilterWithRefConUPP pointer.

`HIMovieViewChangeAttributes` (page 1180)
    Changes the views attributes.

`HIMovieViewCreate` (page 1180)
    Creates an HIMovieView object.

`HIMovieViewGetAttributes` (page 1181)
    Returns the view's current attributes.

`HIMovieViewGetControllerBarSize` (page 1181)
    Returns the size of the visible movie controller bar.

`HIMovieViewGetMovie` (page 1182)
    Returns the view's current movie.

`HIMovieViewGetMovieController` (page 1182)
    Returns the view's current movie controller.

`HIMovieViewPause` (page 1183)
    Pauses the view's current movie.

`HIMovieViewPlay` (page 1183)
    Plays the view's current movie.

`HIMovieViewSetMovie` (page 1184)
    Sets the view's current movie.

`MCAddMovieSegment` (page 1185)
    Undocumented

`MCAdjustCursor` (page 1186)
    Undocumented

`MCGetDoActionsProc` (page 1197)
    Retrieves the DoMCActionProc callback attached to a movie controller.

`MCGetInterfaceElement` (page 1198)
    Gets the interface element of a specified type for a movie controller.

`MCInvalidate` (page 1202)
    Invalidates a region of a movie controller's display.

`MCRemoveAllMovies` (page 1210)
    Removes all movies associated with a controller.

`MCRemoveAMovie` (page 1211)
    Removes one movie from a multi-movie controller.

`MCRemoveMovie` (page 1211)
    Removes a movie from a movie controller.

`MCSetActionFilter` (page 1212)
    Sets the MCActionFilterProc callback for a movie controller.

`MCSetControllerCapabilities` (page 1215)
    Undocumented

`MCTrimMovieSegment` (page 1220)
  Undocumented

`NewMCActionFilterUPP` (page 1222)
  Allocates a Universal Procedure Pointer for the MCActionFilterProc callback.

`NewMCActionFilterWithRefConUPP` (page 1222)
  Allocates a Universal Procedure Pointer for the MCActionFilterWithRefConProc callback.

Functions

DisposeMCActionFilterUPP

Disposes of a MCActionFilterUPP pointer.

```
void DisposeMCActionFilterUPP (
   MCActionFilterUPP userUPP
);
```

Parameters

userUPP
  A `MCActionFilterUPP` pointer. See `Universal Procedure Pointers`.

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

DisposeMCActionFilterWithRefConUPP

Disposes of a MCActionFilterWithRefConUPP pointer.

```
void DisposeMCActionFilterWithRefConUPP (
   MCActionFilterWithRefConUPP userUPP
);
```

Parameters

userUPP
  A `MCActionFilterWithRefConUPP` pointer. See `Universal Procedure Pointers`.

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTKitMovieShuffler

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.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
SimpleHIMovieViewPlayer

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.

Availability
Available in Mac OS X v10.3 and later.

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.

Availability
Available in Mac OS X v10.3 and later.

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.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

SimpleHIMovieViewPlayer

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.

Availability

Available in Mac OS X v10.3 and later.

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.

Availability
Available in Mac OS X v10.3 and later.

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.

Availability
Available in Mac OS X v10.3 and later.

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.

Availability

Available in Mac OS X v10.3 and later.

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.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

SimpleHIMovieViewPlayer

Declared In

`HIMovieView.h`

MCActivate

Lets a controller respond to activate, deactivate, suspend, and resume events.

```
ComponentResult MCActivate (
   MovieController mc,
   WindowRef w,
   Boolean activate
);
```

Parameters

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

w

A pointer to the window in which the event has occurred.

activate

The nature of the `event`. Set this parameter to TRUE for activate and resume events. Set it to FALSE for deactivate and suspend events.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CarbonQTGraphicImport

MakeEffectMovie

qtbigscreen

qtbigscreen.win

QTCarbonShell

Declared In

`Movies.h`

MCAddMovieSegment

Undocumented

```
ComponentResult MCAddMovieSegment (
   MovieController mc,
   Movie srcMovie,
   Boolean scaled
);
```

Parameters

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

srcMovie
: The source movie. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), or `NewMovieFromHandle` (page 1400).

scaled
: *Undocumented*

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

MCAdjustCursor

Undocumented

```
ComponentResult MCAdjustCursor (
   MovieController mc,
   WindowRef w,
   Point where,
   long modifiers
);
```

Parameters

mc
: The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

w
: A pointer to the window in which the cursor is located.

where
: The location of the cursor. This value is expressed in the local coordinates of the window specified by the `w` parameter.

modifiers
: The cursor form (see below). See these constants:

  `kQTCursorOpenHand`

  `kQTCursorClosedHand`

  `kQTCursorPointingHand`

  `kQTCursorRightArrow`

  `kQTCursorLeftArrow`

  `kQTCursorDownArrow`

  `kQTCursorUpArrow`

  `kQTCursorIBeam`

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

MCClear

Removes the current movie selection from the movie associated with a specified controller.

```
ComponentResult MCClear (
   MovieController mc
);
```

Parameters

mc
: The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
MakeEffectMovie

mfc.win

qteffects.win

qtgraphics.win

vrbackbuffer.win

Declared In
`Movies.h`

MCClick

Lets a controller respond when the user clicks in a movie controller window.

```
ComponentResult MCClick (
   MovieController mc,
   WindowRef w,
   Point where,
   long when,
   long modifiers
);
```

Parameters

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

w

A pointer to the window in which the event has occurred.

where

The location of the click. This value is expressed in the local coordinates of the window specified by the `w` parameter. Your application must convert this value from the global coordinates returned in the `EventRecord` structure.

when

Indicates when the user pressed the mouse button. You obtain this value from the `EventRecord` structure.

modifiers

Specifies modifier flags for the event. You obtain this value from the `EventRecord` structure.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

QTCarbonShell

Declared In

`Movies.h`

MCCopy

Returns a copy of the current movie selection from the movie associated with a specified controller.

```
Movie MCCopy (
   MovieController mc
);
```

Parameters

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

Return Value
A copy of the movie.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
MakeEffectMovie

mfc.win

qteffects.win

qtgraphics.win

vrbackbuffer.win

Declared In
`Movies.h`

MCCut

Returns a copy of the current movie selection from the movie associated with a specified controller and then removes the current movie selection from the source movie.

```
Movie MCCut (
   MovieController mc
);
```

Parameters
mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

Return Value
A copy of the current movie selection.

Discussion
Your application is responsible for the returned movie. `MCCut` returns a movie containing the current selection from the movie associated with the specified controller. If the user has not made a selection, the returned movie reference is set to `NIL`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
MakeEffectMovie

mfc.win

qteffects.win

qtgraphics.win

vrbackbuffer.win

Declared In

`Movies.h`

MCDoAction

Invokes a movie controller component and makes it perform a specified action.

```
ComponentResult MCDoAction (
   MovieController mc,
   short action,
   void *params
);
```

Parameters

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

action

The action to be taken. See `Movie Controller Actions`.

params

A pointer to the parameter data appropriate to the action. See `Movie Controller Actions` for information about the parameters required for each supported action.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

MakeEffectMovie

qtcontroller

qtcontroller.win

qtinfo

qtstreamsplicer.win

Declared In

`Movies.h`

MCDraw

Responds to an update event.

```
ComponentResult MCDraw (
   MovieController mc,
   WindowRef w
);
```

Parameters

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

w

A pointer to the window in which the update event has occurred.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CarbonQTGraphicImport

QTCarbonShell

SimpleVideoOut

Declared In

`Movies.h`

MCDrawBadge

Displays a controller's badge.

```
ComponentResult MCDrawBadge (
   MovieController mc,
   RgnHandle movieRgn,
   RgnHandle *badgeRgn
);
```

Parameters

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

movieRgn

The boundary region of the controller's movie.

badgeRgn

A pointer to a region that is to receive information about the location of the badge. The movie controller returns the region where the badge is displayed. If you are not interested in this information, you may set this parameter to `NIL`. Your application must dispose of this handle.

Return Value
You can access Movie Toolbox error returns through GetMoviesError (page 221) and GetMoviesStickyError (page 222), as well as in the function result. See Error Codes.

Discussion
This function places the badge in an appropriate location based on the location of the controller's movie. MCDrawBadge can be useful in circumstances where you are using a movie controller component but do not want to incur the overhead of having the QuickTime movie in memory all the time. This function allows you to display the badge without having to display the movie. In addition, you can use the badge region to perform mouse-down event testing.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
Movies.h

MCEnableEditing

Enables and disables editing for a movie controller.

```
ComponentResult MCEnableEditing (
   MovieController mc,
   Boolean enabled
);
```

Parameters

mc
The movie controller for the operation. You obtain this identifier from OpenComponent or OpenDefaultComponent, or from NewMovieController (page 1392).

enabled
Specifies whether to enable or disable editing for the controller. Set this parameter to TRUE to enable editing; set it to FALSE to disable editing.

Return Value
You can access Movie Toolbox error returns through GetMoviesError (page 221) and GetMoviesStickyError (page 222), as well as in the function result. See Error Codes.

Discussion
Once editing is enabled for a controller, the user may edit the movie associated with the controller.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
MakeEffectMovie
mfc.win
qteffects.win

qtstreamsplicer
qtstreamsplicer.win

Declared In
`Movies.h`

MCGetClip

Obtains information describing a movie controller's clipping regions.

```
ComponentResult MCGetClip (
   MovieController mc,
   RgnHandle *theClip,
   RgnHandle *movieClip
);
```

Parameters

mc
The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

theClip
A pointer to a field that is to receive a handle to the clipping region of the entire movie controller. You must dispose of this region when you are done with it. If you are not interested in this information, set this parameter to `NIL`.

movieClip
A pointer to a field that is to receive a handle to the clipping region of the controller's movie. You must dispose of this region when you are done with it. If you are not interested in this information, set this parameter to `NIL`.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

MCGetControllerBoundsRect

Returns a movie controller's boundary rectangle.

```
ComponentResult MCGetControllerBoundsRect (
   MovieController mc,
   Rect *bounds
);
```

Parameters

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

bounds

A pointer to a `Rect` structure that is to receive the coordinates of the movie controller's boundary rectangle. If there is insufficient screen space to display the controller, the function may return an empty structure.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

MakeEffectMovie

qtbigscreen

qtbigscreen.win

vrscript

vrscript.win

Declared In

`Movies.h`

MCGetControllerBoundsRgn

Returns the actual region occupied by the controller and its movie.

```
RgnHandle MCGetControllerBoundsRgn (
   MovieController mc
);
```

Parameters

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

Return Value

A handle to a `MacRegion` structure that reflects the size, shape, and location of the controller. Your application must dispose of this structure.

Discussion

As with `MCGetControllerBoundsRect` (page 1193), this function returns a region even if the controller is hidden. Some movie controllers may not be rectangular in shape. If the movie is not attached to its controller, the boundary region encloses only the control portion of the controller.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

QTCarbonShell

Declared In

`Movies.h`

MCGetControllerInfo

Determines the current status of a movie controller and its associated movie, for menu highlighting.

```
ComponentResult MCGetControllerInfo (
   MovieController mc,
   long *someFlags
);
```

Parameters

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

someFlags

A pointer to flags (see below) that specify the current status and capabilities of the controller. More than one flag may be set to 1. See these constants:

`mcInfoUndoAvailable`

`mcInfoCutAvailable`

`mcInfoCopyAvailable`

`mcInfoPasteAvailable`

`mcInfoClearAvailable`

`mcInfoHasSound`

`mcInfoIsPlaying`

`mcInfoIsLooping`

`mcInfoIsInPalindrome`

`mcInfoEditingEnabled`

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

You can use the information returned by this function to control your application's menu highlighting.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
MakeEffectMovie
qtbigscreen
qtbigscreen.win
qteffects.win
qtgraphics.win

Declared In
`Movies.h`

MCGetControllerPort

Returns a movie controller's color graphics port.

```
CGrafPtr MCGetControllerPort (
   MovieController mc
);
```

Parameters

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

Return Value
A pointer to the movie controller's `CGrafPort` structure.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CompressMovies
DigitizerShell
DragAndDrop Shell
mdiplayer.win
MovieGWorlds

Declared In
`Movies.h`

MCGetCurrentTime

Obtains the time value represented by the indicator on the movie controller's slider.

```
TimeValue MCGetCurrentTime (
   MovieController mc,
   TimeScale *scale
);
```

Parameters

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

scale

A pointer to a field that is to receive the time scale for the controller.

Return Value

A time value containing the time shown by the indicator on the movie controller's slider.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

MCGetDoActionsProc

Retrieves the DoMCActionProc callback attached to a movie controller.

```
ComponentResult MCGetDoActionsProc (
   MovieController mc,
   DoMCActionUPP *doMCActionProc,
   long *doMCActionRefCon
);
```

Parameters

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

doMCActionProc

A pointer to a `DoMCActionProc` callback.

doMCActionRefCon

A reference constant that is passed to your callback. This parameter may point to a data structure containing information your function needs.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

MCGetIndMovie

Lets your application to retrieve the movie reference for a movie that is associated with a movie controller.

```
Movie MCGetIndMovie (
   MovieController mc,
   short index
);
```

Parameters

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

index

Index for the movie. When set to 0, this call duplicates the action of the previous call to this function.

Return Value
The movie identifier for the movie that is assigned to the specified controller, or `NIL` if there is no movie assigned to the controller.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

MCGetInterfaceElement

Gets the interface element of a specified type for a movie controller.

```
ComponentResult MCGetInterfaceElement (
   MovieController mc,
   MCInterfaceElement whichElement,
   void *element
);
```

Parameters

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

whichElement

A constant (see below) that identifies the interface element type. See these constants:

- `kMCIEEnabledButtonPicture`
- `kMCIEDisabledButtonPicture`
- `kMCIEDepressedButtonPicture`
- `kMCIEEnabledSizeBoxPicture`
- `kMCIEDisabledSizeBoxPicture`
- `kMCIEEnabledUnavailableButtonPicture`
- `kMCIEDisabledUnavailableButtonPicture`
- `kMCIESoundSlider`
- `kMCIESoundThumb`

element

A pointer to the `element` type.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

MCGetMenuString

Retrieves the text string for a movie controller menu command.

```
ComponentResult MCGetMenuString (
   MovieController mc,
   long modifiers,
   short item,
   Str255 aString
);
```

Parameters

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

modifiers

The current modifiers from the mouse-down or key-down event to which you are responding.

item

One of the movie controller Edit menu constants (see below). See these constants:

- `mcMenuUndo`
- `mcMenuCut`
- `mcMenuCopy`
- `mcMenuPaste`
- `mcMenuClear`

aString

On entry, pass a string of type Str255; on exit, this string is set to the text of the menu item specified by the `item` parameter.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

`MCGetMenuString` is used by `MCSetUpEditMenu` (page 1219).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qtcontroller

qteffects.win

qtgraphics.win

qtwiredactions

vrbackbuffer.win

Declared In

`Movies.h`

MCGetVisible

Returns a value that indicates whether or not a movie controller is visible.

```
ComponentResult MCGetVisible (
   MovieController mc
);
```

Parameters

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

Return Value

If the controller is visible, the function result is set to 1. If the controller is not showing, the function result is set to 0. You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
MakeEffectMovie

qtbigscreen

qtbigscreen.win

vrscript

vrscript.win

Declared In
`Movies.h`

MCGetWindowRgn

Determines the window region that is actually in use by a controller and its movie.

```
RgnHandle MCGetWindowRgn (
   MovieController mc,
   WindowRef w
);
```

Parameters

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

w

A pointer to the window in which the movie controller and its movie are displayed, if the control portion of the controller is attached to the movie. If the controller is detached and in a separate window from the movie, specify one of the windows.

Return Value
A handle to the `MacRegion` structure for the window that is actually in use. Your application must dispose of this structure.

Discussion
The region returned by this function contains only the visible portions of the controller and its movie.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
DigitizerShell

MCComponent

MovieBrowser

MovieGWorlds

vrscript.win

Declared In

`Movies.h`

MCIdle

Performs idle processing for a movie controller.

```
ComponentResult MCIdle (
   MovieController mc
);
```

Parameters

`mc`

The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CarbonQTGraphicImport

MakeEffectMovie

MovieBrowser

qtshellCEvents.win

vrscript.win

Declared In

`Movies.h`

MCInvalidate

Invalidates a region of a movie controller's display.

```
ComponentResult MCInvalidate (
   MovieController mc,
   WindowRef w,
   RgnHandle invalidRgn
);
```

Parameters

`mc`

The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

w

A pointer to the window in which the movie controller and its movie are displayed, if the control portion of the controller is attached to the movie. If the controller is detached and in a separate window from the movie, specify one of the windows.

invalidRgn

A handle to a `MacRegion` structure that defines a region to invalidate.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

QTCarbonShell

vrscript

vrscript.win

Declared In

`Movies.h`

MCIsControllerAttached

Returns a value that indicates whether a movie controller is attached to its movie.

```
ComponentResult MCIsControllerAttached (
   MovieController mc
);
```

Parameters

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

Return Value

If the controller is attached, the returned value is set to 1. If the controller is not attached, the returned value is set to 0. You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

MakeEffectMovie

qtbigscreen

qtbigscreen.win

qtcontroller

qtcontroller.win

Declared In
`Movies.h`

MCIsEditingEnabled

Determines whether editing is currently enabled for a movie controller.

```
long MCIsEditingEnabled (
   MovieController mc
);
```

Parameters

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

Return Value
Returns 1 if editing is enabled; set to 0 if editing is disabled or if the controller component does not support editing.

Discussion
Once editing is enabled for a controller, the user may edit the movie associated with the controller.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

MCIsPlayerEvent

Handles all events for a movie controller.

```
ComponentResult ADD_MEDIA_BASENAME() MCIsPlayerEvent
```

Parameters

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

e

A pointer to the current `EventRecord` structure.

Return Value
A long integer indicating whether the movie controller component handled the event. The component sets this long integer to 1 if it handled the event. Your application should then skip the rest of its event loop and wait for the next event. The return value is 0 otherwise. Your application must then handle the event as part of its normal event processing.

Discussion

The movie controller component does everything necessary to support the movie controller and its associated movie. For example, the component calls `MoviesTask` (page 257) for each movie. The movie controller component also handles suspend and resume events. It treats suspend events as deactivate requests and resume events as activate requests.

The following sample code shows how to convert Windows messages to Macintosh events and then pass those events to the QuickTime movie controller, using this function:

```
// MCIsPlayerEvent coding example
// See "Discovering QuickTime," page 240
MovieController   mc;                   // Movie controller for movie
LRESULT CALLBACK WndProc
        (HWND       hwnd,               // Handle to window
         UINT       iMsg,               // Message type
         WPARAM     wParam,             // Message-dependent parameter
         LPARAM     lParam)             // Message-dependent parameter
{
    MSG             msg;                // Windows message structure
    EventRecord     er;                 // Macintosh event record
    DWORD           dwPos;              // Mouse coordinates of message
    msg.hwnd        =hwnd;             // Window handle
    msg.message     =iMsg;             // Message type
    msg.wParam      =wParam;           // Word-length parameter
    msg.lParam      =lParam;           // Long-word parameter

    msg.time =GetMessageTime();        // Get time of message
    dwPos  =GetMessagePos();           // Get mouse position
    msg.pt.x =LOWORD(dwPos);           // Extract x coordinate
    msg.pt.y =HIWORD(dwPos);           // Extract y coordinate

    WinEventToMacEvent(&msg, &er);      // Convert to event
    MCIsPlayerEvent(mc, &er);           // Pass event to QuickTime

    switch (iMsg) {                     // Dispatch on message type

        . . .       // Handle message according to type

    }  // end switch (iMsg)

}  // end WndProc
```

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

MakeEffectMovie

makeeffectslideshow

qtbigscreen

qtcontroller

qtwiredactions

Declared In
`Movies.h`

MCKey

Handles keyboard events for a movie controller.

```
ComponentResult MCKey (
   MovieController mc,
   SInt8 key,
   long modifiers
);
```

Parameters

mc
The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

key
The keystroke. You obtain this value from the event structure.

modifiers
Modifier flags for the event. You obtain this value from the `EventRecord` structure.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

MCMovieChanged

Informs a movie controller component that your application has used the Movie Toolbox to change the characteristics of its associated movie.

```
ComponentResult MCMovieChanged (
   MovieController mc,
   Movie m
);
```

Parameters

mc
The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

m
The movie that has been changed.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qtinfo

qtinfo.win

qttext

qttext.win

SimpleVideoOut

Declared In
`Movies.h`

MCNewAttachedController

Associates a specified movie with a movie controller.

```
ComponentResult MCNewAttachedController (
   MovieController mc,
   Movie theMovie,
   WindowRef w,
   Point where
);
```

Parameters

mc
: The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

theMovie
: The movie to be associated with the movie controller.

w
: A pointer to the window in which the movie is to be displayed. The movie controller component sets the movie's graphics world to match this window. If you set the `w` parameter to `NIL`, the component uses the current window.

where
: The upper-left corner of the movie within the window specified by the `w` parameter. The movie controller component uses the movie's boundary `Rect` structure to determine the size of the movie. `GetMovieBox` (page 207) returns this structure.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

MCPaste

Inserts a specified movie at the current movie time in the movie associated with a specified controller.

```
ComponentResult MCPaste (
   MovieController mc,
   Movie srcMovie
);
```

Parameters

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

srcMovie

The movie to be inserted into the current selection in the movie associated with the movie controller specified by the `mc` parameter. If you set this parameter to `NIL`, the movie controller component retrieves the source movie from the scrap.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
MakeEffectMovie

mfc.win

qteffects.win

qtgraphics.win

vrbackbuffer.win

Declared In
`Movies.h`

MCPositionController

Controls the position of a movie and its controller on the computer display.

```
ComponentResult MCPositionController (
   MovieController mc,
   const Rect *movieRect,
   const Rect *controllerRect,
   long someFlags
);
```

Parameters

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

movieRect

A pointer to a `Rect` structure that specifies the coordinates of the movie's boundary `Rect` structure.

controllerRect

A pointer to a `Rect` structure that specifies the coordinates of the controller's boundary `Rect` structure. The movie controller component always centers the control portion of the controller inside this rectangle. The movie controller component only uses this parameter when the control portion of the controller is detached from the movie. If you are working with an attached controller, you can set this parameter to `NIL`.

someFlags

Flags (see below) that control how the movie is drawn. If you set these flags to 0, the movie controller component centers the movie in the rectangle specified by `movieRect` and scales the movie to fit in that rectangle. See these constants:

`mcTopLeftMovie`

`mcScaleMovieToFit`

`mcPositionDontInvalidate`

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

MakeEffectMovie

qteffects.win

qtgraphics.win

qtwiredactions

vrbackbuffer.win

Declared In

`Movies.h`

MCPtInController

Reports whether a point is in the control area of a movie.

```
ComponentResult MCPtInController (
   MovieController mc,
   Point thePt,
   Boolean *inController
);
```

Parameters

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

thePt

The point to be checked. This point must be passed in local coordinates to the controller's window. This point is checked only against the movie controller's controls, not the movie itself.

inController

Returns true if the point is in the controller; false if it is not.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

While you could always determine if a point is contained in a movie, using `PtInMovie` (page 1634), the `MCPtInController` function allows you to determine if a point is in the control area of a movie.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

MCRemoveAllMovies

Removes all movies associated with a controller.

```
ComponentResult MCRemoveAllMovies (
   MovieController mc
);
```

Parameters

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

MCRemoveAMovie

Removes one movie from a multi-movie controller.

```
ComponentResult MCRemoveAMovie (
   MovieController mc,
   Movie m
);
```

Parameters

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

m

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), or `NewMovieFromHandle` (page 1400).

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

MCRemoveMovie

Removes a movie from a movie controller.

```
ComponentResult MCRemoveMovie (
   MovieController mc
);
```

Parameters

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

MCSetActionFilter

Sets the MCActionFilterProc callback for a movie controller.

```
ComponentResult MCSetActionFilter (
   MovieController mc,
   MCActionFilterUPP blob
);
```

Parameters

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

blob

A Universal Procedure Pointer to an `MCActionFilterProc` callback.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

MCSetActionFilterWithRefCon

Establishes an action filter function for a movie controller.

```
ComponentResult MCSetActionFilterWithRefCon (
   MovieController mc,
   MCActionFilterWithRefConUPP blob,
   long refCon
);
```

Parameters

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

blob

A pointer to your `MCActionFilterWithRefConProc` callback. Set this parameter to `NIL` to remove an existing callback.

refCon

A reference constant value. The movie controller component passes this reference constant to your action filter callback each time it calls it. Use this parameter to point to a data structure containing any information your callback needs.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

The movie controller component calls your action filter function each time the component receives an action for its movie controller. Your filter function is then free to handle the action or to refer it back to the movie controller component. If you refer it back to the movie controller component, the component handles the action.

If your filter function handles an action, you can handle the action in any way you desire. For example, your filter function could change the operation of movie controller buttons. More commonly, applications use the action filter function to monitor actions of the controller. For instance, your filter function might enable you to find out when the user clicks the play button, so that your application can enable appropriate menu selections. Alternatively, you can use the filter function to detect when the user resizes the movie.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

MakeEffectMovie

qteffects.win

qtgraphics.win

QTKitMovieShuffler

vrbackbuffer.win

Declared In

`Movies.h`

MCSetClip

Lets you set a movie controller's clipping region.

```
ComponentResult MCSetClip (
   MovieController mc,
   RgnHandle theClip,
   RgnHandle movieClip
);
```

Parameters

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

theClip

A handle to a region that defines the controller's clipping region. This clipping region affects the entire movie controller and its movie, including the controller's badge and associated controls. Set this parameter to `NIL` to clear the controller's clipping region.

movieClip

A handle to a region that defines the clipping region of the controller's movie. This clipping region affects only the movie and the badge, not the movie controller. Set this parameter to `NIL` to clear the movie clipping region.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

MCSetControllerAttached

Lets your application control whether a movie controller is attached to its movie or detached from it.

```
ComponentResult MCSetControllerAttached (
   MovieController mc,
   Boolean attach
);
```

Parameters

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

attach

The action for this function. Set the `attach` parameter to TRUE to cause the controller to be attached to its movie. Set this parameter to FALSE to detach the controller from its movie.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

MakeEffectMovie

qtbigscreen

qtbigscreen.win

qtskins.win

SimpleVideoOut

Declared In
`Movies.h`

MCSetControllerBoundsRect

Lets you change the position and size of a movie controller.

```
ComponentResult MCSetControllerBoundsRect (
   MovieController mc,
   const Rect *bounds
);
```

Parameters

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

bounds

A pointer to a `Rect` structure that contains the new boundary `Rect` structure for the movie controller.

Return Value
See `Error Codes`. Returns a value of `controllerBoundsNotExact` if the boundary rectangle has been changed but does not correspond to the rectangle you specified. In this case, the new boundary rectangle is always smaller than the requested rectangle. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CompressMovies

DigitizerShell

DragAndDrop Shell

MovieGWorlds

qtbigscreen

Declared In
`Movies.h`

MCSetControllerCapabilities

Undocumented

```
ComponentResult MCSetControllerCapabilities (
   MovieController mc,
   long flags,
   long flagsMask
);
```

Parameters

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

flags

Undocumented

flagsMask

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 6.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`Movies.h`

MCSetControllerPort

Lets your application set the graphics port for a movie controller.

```
ComponentResult MCSetControllerPort (
   MovieController mc,
   CGrafPtr gp
);
```

Parameters

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

gp

A pointer to the new graphics port for the movie controller. Set this parameter to `NIL` to use the current graphics port.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

Movie controller components use `MCSetControllerPort` each time you create a new movie controller. Hence, your component must be set to a valid port before creating a new movie controller. You can use this function to place a movie and its associated movie controller in different graphics ports. If you are using an attached controller, both the controller and the movie's graphics ports are changed. If you are using a detached controller, this function changes only the graphics port of the control portion of the controller. You must use `SetMovieGWorld` (page 290) followed by `MCMovieChanged` (page 1206) to change other portions.

```
pascal ComponentResult MCSetControllerPort (MovieController mc,
                                            CGrafPtr gp);
```

Special Considerations

The movie controller component may use the foreground and background colors from the graphics port at the time this function is called to colorize the movie controller.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

MovieGWorlds

qtbigscreen

QTCarbonShell

SimpleVideoOut

vrscript.win

Declared In

`Movies.h`

MCSetDuration

Lets your application set a controller's duration in the case where a controller does not have a movie associated with it.

```
ComponentResult MCSetDuration (
   MovieController mc,
   TimeValue duration
);
```

Parameters

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

duration

The new duration for the movie. This duration value must be in the controller's time scale.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

MCSetIdleManager

Lets a movie controller component report its idling needs.

```
ComponentResult MCSetIdleManager (
   MovieController mc,
   IdleManager im
);
```

Parameters

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

im

A pointer to an opaque data structure that belongs to the Mac OS Idle Manager. You get this pointer by calling `QTIdleManagerOpen` (page 273).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`Movies.h`

MCSetMovie

Associates a movie with a specified movie controller.

```
ComponentResult MCSetMovie (
   MovieController mc,
   Movie theMovie,
   WindowRef movieWindow,
   Point where
);
```

Parameters

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

theMovie

The movie to be associated with the movie controller. Set this value to `NIL` to remove the movie from the controller.

movieWindow
: The window in which the movie is to be displayed. The movie controller component sets the movie's graphics world to match this window. If you set the `w` parameter to `NIL`, the component uses the current window.

where
: The upper-left corner of the movie within the window specified by the `movieWindow` parameter. The movie controller component uses the movie's boundary `Rect` structure to determine the size of the movie. `GetMovieBox` (page 207) returns this structure.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
MovieBrowser

Declared In
`Movies.h`

MCSetUpEditMenu

Correctly highlights and names the items in your application's Edit menu.

```
ComponentResult MCSetUpEditMenu (
   MovieController mc,
   long modifiers,
   MenuRef mh
);
```

Parameters

mc
: The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

modifiers
: The current modifiers from the mouse-down or key-down event to which you are responding.

mh
: A menu handler for your current Edit menu. The first six items in your Edit menu should be the standard editing commands: Undo, a blank line, Cut, Copy, Paste, and Clear.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code

qtcontroller

qteffects.win

qtgraphics.win

qtwiredactions

vrbackbuffer.win

Declared In

`Movies.h`

MCSetVisible

Lets your application control the visibility of a movie controller.

```
ComponentResult MCSetVisible (
   MovieController mc,
   Boolean visible
);
```

Parameters

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

visible

Set to TRUE to cause the controller to be visible, or FALSE to make the controller invisible.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

MakeEffectMovie

qtbigscreen

qtbigscreen.win

QTCarbonShell

watchme

Declared In

`Movies.h`

MCTrimMovieSegment

Undocumented

```
ComponentResult MCTrimMovieSegment (
   MovieController mc
);
```

Parameters

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

MCUndo

Lets your application discard the effects of the most recent edit operation.

```
ComponentResult MCUndo (
   MovieController mc
);
```

Parameters

mc

The movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

MakeEffectMovie

mfc.win

qteffects.win

qtgraphics.win

vrbackbuffer.win

Declared In

`Movies.h`

NewMCActionFilterUPP

Allocates a Universal Procedure Pointer for the MCActionFilterProc callback.

```
MCActionFilterUPP NewMCActionFilterUPP (
   MCActionFilterProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your application-defined function.

Return Value
A new UPP; see `Universal Procedure Pointers`.

Discussion
This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes
Introduced in QuickTime 4.1. Replaces `NewMCActionFilterProc`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

NewMCActionFilterWithRefConUPP

Allocates a Universal Procedure Pointer for the MCActionFilterWithRefConProc callback.

```
MCActionFilterWithRefConUPP NewMCActionFilterWithRefConUPP (
   MCActionFilterWithRefConProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your application-defined function.

Return Value
A new UPP; see `Universal Procedure Pointers`.

Discussion
This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes
Introduced in QuickTime 4.1. Replaces `NewMCActionFilterWithRefConProc`.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qtcontroller
qteffects.win
qtgraphics.win
qtwiredactions

vrbackbuffer.win

Declared In
`Movies.h`

Callbacks

MCActionFilterProc

Responds to movie controller actions.

```
typedef Boolean (*MCActionFilterProcPtr) (MovieController mc, short *action, void
 *params);
```

If you name your function `MyMCActionFilterProc`, you would declare it this way:

```
Boolean MyMCActionFilterProc (
    MovieController    mc,
    short              *action,
    void               *params );
```

Parameters

mc
Specifies the movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

action
A movie controller action. For a list of actions, see Chapter 2 of *Inside Macintosh: QuickTime* Components.

params
A pointer to a structure, such as `QTStatusStringRecord` or `ResolvedQTEventSpec`, that passes information to the callback. See `Movies.h`.

Return Value
Undocumented

Discussion
Movie controller components allow your application to field movie controller actions. You define an `MCActionFilterProc` in your application and assign it to a controller by calling the `MCSetActionFilterWithRefCon` function.

Declared In
`Movies.h, HIMovieView.h`

MCActionFilterWithRefConProc

Responds to movie controller actions with a reference constant.

```
typedef Boolean (*MCActionFilterWithRefConProcPtr) (MovieController mc, short
action, void *params, long refCon);
```

If you name your function `MyMCActionFilterWithRefConProc`, you would declare it this way:

```
Boolean MyMCActionFilterWithRefConProc (
    MovieController    mc,
    short              action,
    void               *params,
    long               refCon );
```

Parameters

mc

Specifies the movie controller for the operation. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

action

A movie controller action. For a list of actions, see Chapter 2 of *Inside Macintosh: QuickTime* Components.

params

A pointer to a structure, such as `QTStatusStringRecord`, that passes information to the callback. See `Movies.h`.

refCon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

Return Value

Undocumented

Declared In

`Movies.h, HIMovieView.h`

Data Types

MCActionFilterUPP

Represents a type used by the Movie Controller API.

```
typedef STACK_UPP_TYPE(MCActionFilterProcPtr) MCActionFilterUPP;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

MCActionFilterWithRefConUPP

Represents a type used by the Movie Controller API.

```
typedef STACK_UPP_TYPE(MCActionFilterWithRefConProcPtr) MCActionFilterWithRefConUPP;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

MCInterfaceElement

Represents a type used by the Movie Controller API.

```
typedef unsigned long MCInterfaceElement;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

OptionBits

Represents a type used by the Movie Controller API.

```
typedef UInt32 OptionBits;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OSTypes.h`

Constants

Movie Controller Options

Constants that represent options for movie controllers.

```
enum {
  kMCIEEnabledButtonPicture     = 1,
  kMCIEDisabledButtonPicture    = 2,
  kMCIEDepressedButtonPicture   = 3,
  kMCIEEnabledSizeBoxPicture    = 4,
  kMCIEDisabledSizeBoxPicture   = 5,
  kMCIEEnabledUnavailableButtonPicture = 6,
  kMCIEDisabledUnavailableButtonPicture = 7,
  kMCIESoundSlider              = 128,
  kMCIESoundThumb               = 129,
  kMCIEColorTable               = 256,
  kMCIEIsFlatAppearance         = 257,
  kMCIEDoButtonIconsDropOnDepress = 258
};
enum {
  mcFlagSuppressMovieFrame      = 1 << 0,
  mcFlagSuppressStepButtons     = 1 << 1,
  mcFlagSuppressSpeakerButton   = 1 << 2,
  mcFlagsUseWindowPalette       = 1 << 3,
  mcFlagsDontInvalidate         = 1 << 4,
  mcFlagsUseCustomButton        = 1 << 5
};
enum {
  mcInfoUndoAvailable           = 1 << 0,
  mcInfoCutAvailable            = 1 << 1,
  mcInfoCopyAvailable           = 1 << 2,
  mcInfoPasteAvailable          = 1 << 3,
  mcInfoClearAvailable          = 1 << 4,
  mcInfoHasSound                = 1 << 5,
  mcInfoIsPlaying               = 1 << 6,
  mcInfoIsLooping               = 1 << 7,
  mcInfoIsInPalindrome          = 1 << 8,
  mcInfoEditingEnabled          = 1 << 9,
  mcInfoMovieIsInteractive      = 1 << 10
};
```

Constants

`kMCIESoundThumb`

The indicator on the sound slider.

Available in Mac OS X v10.0 and later.

Declared in `Movies.h`.

`mcFlagSuppressMovieFrame`

If this flag is set to 1, the controller does not display a frame around the movie. By default, this flag is set to 0.

Available in Mac OS X v10.0 and later.

Declared in `Movies.h`.

`mcFlagSuppressStepButtons`

If this flag is set to 1, the controller does not display the step buttons. By default, this flag is set to 0.

Available in Mac OS X v10.0 and later.

Declared in `Movies.h`.

`mcFlagSuppressSpeakerButton`
: If this flag is set to 1, the controller does not display the speaker button. By default, this flag is set to 0.

  Available in Mac OS X v10.0 and later.

  Declared in `Movies.h`.

`mcFlagsUseWindowPalette`
: If this flag is set to 1, the movie controller does not manage the window palette. This ensures that a movie's colors are reproduced as accurately as possible. This flag is particularly useful for movies with custom color tables. By default, this flag is set to 0.

  Available in Mac OS X v10.0 and later.

  Declared in `Movies.h`.

Declared In
`Movies.h, HIMovieView.h`

MCAdjustCursor Values

Constants passed to MCAdjustCursor.

```
enum {
  kQTCursorOpenHand              = -19183,
  kQTCursorClosedHand            = -19182,
  kQTCursorPointingHand          = -19181,
  kQTCursorRightArrow            = -19180,
  kQTCursorLeftArrow             = -19179,
  kQTCursorDownArrow             = -19178,
  kQTCursorUpArrow               = -19177,
  kQTCursorIBeam                 = -19176
};
```

Declared In
`Movies.h, HIMovieView.h`

MCGetMenuString Values

Constants passed to MCGetMenuString.

```
enum {
  mcMenuUndo                     = 1,
  mcMenuCut                      = 3,
  mcMenuCopy                     = 4,
  mcMenuPaste                    = 5,
  mcMenuClear                    = 6
};
```

Declared In
`Movies.h, HIMovieView.h`

MCPositionController Values

Constants passed to MCPositionController.

```
enum {
  mcPositionDontInvalidate      = 1 << 5
};
```

Declared In

`Movies.h`, `HIMovieView.h`

QuickTime Movie Properties Reference

Framework:	Frameworks/QuickTime.framework
Declared in	Movies.h

Overview

QuickTime movies and movie tracks have properties that an application can manage, including embedded metadata and sample tables that determine what, how, and when the movie will present its data.

Functions by Task

Working With QuickTime Metadata

`QTCopyMediaMetaData` (page 1239)
: Retains a media's metadata object and returns it.

`QTCopyMovieMetaData` (page 1239)
: Retains a movie's metadata object and returns it.

`QTCopyTrackMetaData` (page 1240)
: Retains a track's metadata object and returns it.

`QTMetaDataAddItem` (page 1244)
: Adds an inline metadata item to the metadata storage format.

`QTMetaDataGetItemProperty` (page 1246)
: Returns a property of a metadata item.

`QTMetaDataGetItemPropertyInfo` (page 1247)
: Returns information about a property of a metadata item.

`QTMetaDataGetItemValue` (page 1248)
: Returns the value of a metadata item from an item identifier.

`QTMetaDataGetNextItem` (page 1248)
: Returns the next metadata item corresponding to a specified key.

`QTMetaDataGetProperty` (page 1250)
: Returns a property of a metadata object.

`QTMetaDataGetPropertyInfo` (page 1250)
: Returns information about a property of a metadata object.

`QTMetaDataRelease` (page 1251)
: Decrements the retain count of a metadata object.

QTMetaDataRemoveItem (page 1252)
: Removes a metadata item from a storage format.

QTMetaDataRemoveItemsWithKey (page 1252)
: Removes metadata items with a specific key from the storage format.

QTMetaDataRetain (page 1253)
: Increments the retain count of a metadata object.

QTMetaDataSetItem (page 1254)
: Sets the value of the metadata item from the item identifier.

QTMetaDataSetItemProperty (page 1254)
: Sets a property of a metadata item.

QTMetaDataSetProperty (page 1255)
: Sets a property of a metadata object.

Working With QuickTime Sample Tables

QTSampleTableAddSampleDescription (page 1257)
: Adds a sample description to a sample table, returning a sample description ID that can be used to refer to it.

QTSampleTableAddSampleReferences (page 1258)
: Adds sample references to a sample table.

QTSampleTableCopySampleDescription (page 1259)
: Retrieves a sample description from a sample table.

QTSampleTableCreateMutable (page 1260)
: Creates a new, empty sample table.

QTSampleTableCreateMutableCopy (page 1261)
: Copies a sample table.

QTSampleTableGetDataOffset (page 1261)
: Returns the data offset of a sample.

QTSampleTableGetDataSizePerSample (page 1262)
: Returns the data size of a sample.

QTSampleTableGetDecodeDuration (page 1262)
: Returns the decode duration of a sample.

QTSampleTableGetDisplayOffset (page 1263)
: Returns the offset from decode time to display time of a sample.

QTSampleTableGetNextAttributeChange (page 1263)
: Finds the next sample number at which one or more of a set of given sample attributes change.

QTSampleTableGetNumberOfSamples (page 1265)
: Returns the number of samples in a sample table.

QTSampleTableGetProperty (page 1265)
: Returns the value of a specific sample table property.

QTSampleTableGetPropertyInfo (page 1266)
: Returns information about the properties of a sample table.

QTSampleTableGetSampleDescriptionID (page 1268)
: Returns the sample description ID of a sample.

QTSampleTableGetSampleFlags (page 1268)
Returns the media sample flags of a sample.

QTSampleTableGetTimeScale (page 1269)
Returns the timescale of a sample table.

QTSampleTableGetTypeID (page 1269)
Returns the CFTypeID value for the current sample table.

QTSampleTableRelease (page 1269)
Decrements the retain count of a sample table.

QTSampleTableReplaceRange (page 1270)
Replaces a range of samples in a sample table with a range of samples from another sample table.

QTSampleTableRetain (page 1271)
Increments the retain count of a sample table.

QTSampleTableSetProperty (page 1271)
Sets the value of a specific sample table property.

QTSampleTableSetTimeScale (page 1272)
Changes the timescale of a sample table.

Supporting Functions

DisposeQTTrackPropertyListenerUPP (page 1233)
Disposes a track property listener UPP.

InvokeQTTrackPropertyListenerUPP (page 1233)
Invokes the specified property listener of a track.

MusicMediaGetIndexedTunePlayer (page 1233)
Undocumented

NewQTTrackPropertyListenerUPP (page 1235)
Creates a new callback to monitor a track property.

QTAddMoviePropertyListener (page 1235)
Installs a callback to monitor a movie property.

QTAddTrackPropertyListener (page 1236)
Installs a callback to monitor a track property.

QTGetMovieProperty (page 1241)
Returns the value of a specific movie property.

QTGetMoviePropertyInfo (page 1241)
Returns information about the properties of a movie.

QTGetTrackProperty (page 1242)
Returns the value of a specific track property.

QTGetTrackPropertyInfo (page 1243)
Returns information about the properties of a track.

QTRemoveMoviePropertyListener (page 1256)
Removes a movie property monitoring callback.

QTRemoveTrackPropertyListener (page 1257)
Removes a track property monitoring callback

`QTSetMovieProperty` (page 1274)
: Sets the value of a specific movie property.

`QTSetTrackProperty` (page 1275)
: Sets the value of a specific track property.

`DisposeQTBandwidthNotificationUPP` (page 1232) Deprecated in Mac OS X v10.4
: Disposes of a QTBandwidthNotificationUPP pointer.

`NewQTBandwidthNotificationUPP` (page 1234) Deprecated in Mac OS X v10.4
: Allocates a Universal Procedure Pointer for the QTBandwidthNotificationProc callback.

`QTBandwidthRelease` (page 1236) Deprecated in Mac OS X v10.4
: Undocumented

`QTBandwidthRequest` (page 1237) Deprecated in Mac OS X v10.4
: Undocumented

`QTBandwidthRequestForTimeBase` (page 1238) Deprecated in Mac OS X v10.4
: Undocumented

`QTScheduledBandwidthRelease` (page 1273) Deprecated in Mac OS X v10.4
: Undocumented

`QTScheduledBandwidthRequest` (page 1273) Deprecated in Mac OS X v10.4
: Undocumented

Functions

DisposeQTBandwidthNotificationUPP

Disposes of a QTBandwidthNotificationUPP pointer. (Deprecated in Mac OS X v10.4.)

```
void DisposeQTBandwidthNotificationUPP (
   QTBandwidthNotificationUPP userUPP
);
```

Parameters

userUPP
: A `QTBandwidthNotificationUPP` pointer. See `Universal Procedure Pointers`.

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`Movies.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.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`Movies.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.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`Movies.h`

MusicMediaGetIndexedTunePlayer

Undocumented

```
ComponentResult MusicMediaGetIndexedTunePlayer (
   ComponentInstance ti,
   long sampleDescIndex,
   ComponentInstance *tp
);
```

Parameters

`ti`

Undocumented

`sampleDescIndex`

Undocumented

`tp`

A pointer to a tune player component instance.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

NewQTBandwidthNotificationUPP

Allocates a Universal Procedure Pointer for the QTBandwidthNotificationProc callback. (Deprecated in Mac OS X v10.4.)

```
QTBandwidthNotificationUPP NewQTBandwidthNotificationUPP (
   QTBandwidthNotificationProcPtr userRoutine
);
```

Parameters

`userRoutine`

A pointer to your application-defined function.

Return Value
A new UPP; see `Universal Procedure Pointers`.

Discussion
This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes
Introduced in QuickTime 4.1. Replaces `NewQTBandwidthNotificationProc`.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

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.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`Movies.h`

QTAddMoviePropertyListener

Installs a callback to monitor a movie property.

```
OSErr QTAddMoviePropertyListener (
   Movie inMovie,
   QTPropertyClass inPropClass,
   QTPropertyID inPropID,
   QTMoviePropertyListenerUPP inListenerProc,
   void *inUserData
);
```

Parameters

inMovie

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

inPropClass

A property class.

inPropID

A property ID.

inListenerProc

A Universal Procedure Pointer to a `QTMoviePropertyListenerProc` callback.

inUserData

A pointer to user data that will be passed to the callback.

Return Value

See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 6.4.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
QTAudioExtractionPanel

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.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
QTAudioExtractionPanel

Declared In
`Movies.h`

QTBandwidthRelease

Undocumented (Deprecated in Mac OS X v10.4.)

```
OSErr QTBandwidthRelease (
   QTBandwidthReference bwRef,
   long flags
);
```

Parameters

bwRef

Undocumented

flags

Undocumented

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`Movies.h`

QTBandwidthRequest

Undocumented (Deprecated in Mac OS X v10.4.)

```
OSErr QTBandwidthRequest (
   long priority,
   QTBandwidthNotificationUPP callback,
   const void *refcon,
   QTBandwidthReference *bwRef,
   long flags
);
```

Parameters

priority

Undocumented

callback

A `QTBandwidthNotificationProc` callback.

refcon

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

bwRef

Undocumented

flags

Undocumented

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`Movies.h`

QTBandwidthRequestForTimeBase

Undocumented (Deprecated in Mac OS X v10.4.)

```
OSErr QTBandwidthRequestForTimeBase (
   TimeBase tb,
   long priority,
   QTBandwidthNotificationUPP callback,
   const void *refcon,
   QTBandwidthReference *bwRef,
   long flags
);
```

Parameters

tb
A time base. Your application obtains this time base identifier from `NewTimeBase` (page 261).

priority
Undocumented

callback
A `QTBandwidthNotificationProc` callback.

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

bwRef
Undocumented

flags
Undocumented

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

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` (page 1630) and `GetTrackMedia` (page 1612).

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` (page 1251) 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.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

QTMetadataEditor

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` (page 259), `NewMovieFromProperties` (page 260), `NewMovieFromFile`, and `NewMovieFromHandle` (page 1400).

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` (page 1251) 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.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

QTMetaData

QTMetadataEditor

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` (page 1628) and `GetMovieTrack` (page 1601).

`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` (page 1251) 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.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

QTMetadataEditor

Declared In

`Movies.h`

QTGetMovieProperty

Returns the value of a specific movie property.

```
OSErr QTGetMovieProperty (
   Movie inMovie,
   QTPropertyClass inPropClass,
   QTPropertyID inPropID,
   ByteCount inPropValueSize,
   QTPropertyValuePtr outPropValueAddress,
   ByteCount *outPropValueSizeUsed
);
```

Parameters

inMovie

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

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

See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 6.4.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

QTAudioExtractionPanel

QTExtractAndConvertToAIFF

QTExtractAndConvertToMovieFile

Declared In

`Movies.h`

QTGetMoviePropertyInfo

Returns information about the properties of a movie.

```
OSErr QTGetMoviePropertyInfo (
   Movie inMovie,
   QTPropertyClass inPropClass,
   QTPropertyID inPropID,
   QTPropertyValueType *outPropType,
   ByteCount *outPropValueSize,
   UInt32 *outPropertyFlags
);
```

Parameters

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

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

See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 6.4.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

QTAudioExtractionPanel

QTExtractAndConvertToAIFF

QTExtractAndConvertToMovieFile

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.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
QTAudioExtractionPanel

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.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

QTAudioExtractionPanel

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 = 24With `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`.

Discussion
The data type of the metadata item is assumed to be binary.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
QTMetadataEditor

Declared In
`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 1244) and `QTMetaDataGetNextItem` (page 1248).

`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`.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
QTMetaData
QTMetadataEditor

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 1244) and `QTMetaDataGetNextItem` (page 1248).

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`.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
QTMetaData
QTMetadataEditor

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 1244).

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`.

Discussion
You can use this function to get the value of a metadata item that has a known item identifier.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
QTMetaData

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 1244).

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`.

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`.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

QTMetaData

QTMetadataEditor

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`.

Availability
Available in Mac OS X v10.3 and later.

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`.

Availability
Available in Mac OS X v10.3 and later.

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.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
QTMetaData
QTMetadataEditor

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 1244) and `QTMetaDataGetNextItem` (page 1248).

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`.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
QTMetadataEditor

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`.

Availability

Available in Mac OS X v10.3 and later.

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 1251).

Availability

Available in Mac OS X v10.3 and later.

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 1244) and `QTMetaDataGetNextItem` (page 1248).

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 = 24With `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`.

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`.

Availability

Available in Mac OS X v10.3 and later.

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 1244) and `QTMetaDataGetNextItem` (page 1248).

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`.

Availability

Available in Mac OS X v10.3 and later.

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`.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`Movies.h`

QTRemoveMoviePropertyListener

Removes a movie property monitoring callback.

```
OSErr QTRemoveMoviePropertyListener (
   Movie inMovie,
   QTPropertyClass inPropClass,
   QTPropertyID inPropID,
   QTMoviePropertyListenerUPP inListenerProc,
   void *inUserData
);
```

Parameters

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

inPropClass
A property class.

inPropID
A property ID.

inListenerProc
A Universal Procedure Pointer to a `QTMoviePropertyListenerProc` callback.

inUserData
User data to be passed to the callback.

Return Value
See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 6.4.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
QTAudioExtractionPanel

Declared In
`Movies.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.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
QTAudioExtractionPanel

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.

Availability
Available in Mac OS X v10.3 and later.

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. See these constants:

    `mediaSampleNotSync`

    `mediaSampleShadowSync`

sampleDescriptionID
: The ID of a sample description that has been added to the sample table with `QTSampleTableAddSampleDescription` (page 1257).

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.

Availability

Available in Mac OS X v10.3 and later.

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` (page 1587) 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.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
MovieVideoChart

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.

Availability
Available in Mac OS X v10.3 and later.

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.

Availability
Available in Mac OS X v10.3 and later.

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.

Availability
Available in Mac OS X v10.3 and later.

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 `sampleTable` is NULL or if the sample number is out of range.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
MovieVideoChart

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 `sampleTable` is NULL or if the sample number is out of range.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
MovieVideoChart

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 `sampleTable` is NULL or if the sample number is out of range.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
MovieVideoChart

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. See these constants:

`kQTSampleTableAttribute_DiscontiguousData`

`kQTSampleTableAttribute_DataSizePerSampleChange`

`kQTSampleTableAttribute_DecodeDurationChange`

`kQTSampleTableAttribute_DisplayOffsetChange`

`kQTSampleTableAttribute_SampleDescriptionIDChange`

`kQTSampleTableAttribute_SampleFlagsChange`

`kQTSampleTableAnyAttributeChange`

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.

Availability

Available in Mac OS X v10.3 and later.

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.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

MovieVideoChart

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. See these constants:

`kQTPropertyClass_SampleTable`

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. See these constants:

`kQTSampleTablePropertyID_TotalDecodeDuration`

`kQTSampleTablePropertyID_MinDisplayOffset`

`kQTSampleTablePropertyID_MaxDisplayOffset`

`kQTSampleTablePropertyID_MinRelativeDisplayTime`

`kQTSampleTablePropertyID_MaxRelativeDisplayTime`

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.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

MovieVideoChart

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. See these constants:

`kQTPropertyClass_SampleTable`

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. See these constants:

`kQTSampleTablePropertyID_TotalDecodeDuration`

`kQTSampleTablePropertyID_MinDisplayOffset`

`kQTSampleTablePropertyID_MaxDisplayOffset`

`kQTSampleTablePropertyID_MinRelativeDisplayTime`

`kQTSampleTablePropertyID_MaxRelativeDisplayTime`

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.

Availability

Available in Mac OS X v10.3 and later.

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 `sampleTable` is NULL or if the sample number is out of range.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

MovieVideoChart

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 `sampleTable` is NULL or if the sample number is out of range.

Discussion

This function can return one or more of the following constants:

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

MovieVideoChart

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.

Availability
Available in Mac OS X v10.3 and later.

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`.

Availability
Available in Mac OS X v10.3 and later.

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.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

MovieVideoChart

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.

Availability

Available in Mac OS X v10.3 and later.

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.

Availability

Available in Mac OS X v10.3 and later.

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. See these constants:

`kQTPropertyClass_SampleTable`

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. See these constants:

`kQTSampleTablePropertyID_TotalDecodeDuration`

`kQTSampleTablePropertyID_MinDisplayOffset`

`kQTSampleTablePropertyID_MaxDisplayOffset`

`kQTSampleTablePropertyID_MinRelativeDisplayTime`

`kQTSampleTablePropertyID_MaxRelativeDisplayTime`

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.

Availability

Available in Mac OS X v10.3 and later.

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.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`Movies.h`

QTScheduledBandwidthRelease

Undocumented (Deprecated in Mac OS X v10.4.)

```
OSErr QTScheduledBandwidthRelease (
   QTScheduledBandwidthReference sbwRef,
   long flags
);
```

Parameters

`sbwRef`

A pointer to an opaque data structure.

`flags`

Undocumented

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 4.1.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`Movies.h`

QTScheduledBandwidthRequest

Undocumented (Deprecated in Mac OS X v10.4.)

```
OSErr QTScheduledBandwidthRequest (
   QTScheduledBandwidthPtr scheduleRec,
   QTBandwidthNotificationUPP notificationCallback,
   void *refcon,
   QTScheduledBandwidthReference *sbwRef,
   long flags
);
```

Parameters

scheduleRec
A pointer to a `QTScheduledBandwidthRecord` structure.

notificationCallback
A Universal Procedure Pointer that accesses a `QTBandwidthNotificationProc` callback.

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

sbwRef
A pointer to an opaque data structure.

flags
Undocumented

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`Movies.h`

QTSetMovieProperty

Sets the value of a specific movie property.

```
OSErr QTSetMovieProperty (
   Movie inMovie,
   QTPropertyClass inPropClass,
   QTPropertyID inPropID,
   ByteCount inPropValueSize,
   ConstQTPropertyValuePtr inPropValueAddress
);
```

Parameters

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

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
See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 6.4.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
QTAudioExtractionPanel

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.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
QTAudioExtractionPanel

Declared In
`Movies.h`

Callbacks

QTBandwidthNotificationProc

Undocumented

```
typedef OSErr (*QTBandwidthNotificationProcPtr) (long flags, void *reserved, void
 *refcon);
```

If you name your function `MyQTBandwidthNotificationProc`, you would declare it this way:

```
OSErr MyQTBandwidthNotificationProc (
    long    flags,
    void    *reserved,
    void    *refcon );
```

Parameters

`flags`

Undocumented

`reserved`

Reserved.

`refcon`

Pointer to a reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

Return Value
See `Error Codes`. Your callback should return `noErr` if there is no error.

Declared In
`Movies.h`

Data Types

QTBandwidthNotificationUPP

Represents a type used by the Movie Properties API.

```
typedef STACK_UPP_TYPE(QTBandwidthNotificationProcPtr) QTBandwidthNotificationUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

QTBandwidthReference

Represents a type used by the Movie Properties API.

```
typedef struct OpaqueQTBandwidthReference * QTBandwidthReference;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

QTScheduledBandwidthPtr

Represents a type used by the Movie Properties API.

```
typedef QTScheduledBandwidthRecord * QTScheduledBandwidthPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

QTScheduledBandwidthRecord

Provides information to the QTScheduledBandwidthRequest function.

```
struct QTScheduledBandwidthRecord {
     long             recordSize;
     long             priority;
     long             dataRate;
     CompTimeValue    startTime;
     CompTimeValue    duration;
     TimeScale        scale;
     TimeBase         base;
 };
```

Fields
`recordSize`

Discussion
The number of bytes in this structure.

`priority`

Discussion
Undocumented

`dataRate`

Discussion
The data rate.

`startTime`

Discussion
The bandwidth usage start time.

`duration`

Discussion
`Duration` of bandwidth usage, or 0 if unknown.

`scale`

Discussion
The timescale of the `duration` field.

`base`

Discussion
The time base.

Declared In
`Movies.h`

QTScheduledBandwidthReference

Represents a type used by the Movie Properties API.

```
typedef struct OpaqueQTScheduledBandwidthReference * QTScheduledBandwidthReference;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

Constants

kQTPropertyClass_SampleTable

Constants grouped with kQTPropertyClass_SampleTable.

```
enum {
  /*
   * Property class for sample tables.
   */
  kQTPropertyClass_SampleTable  = 'qtst',
  /*
   * The total decode duration of all samples in the sample table.
   * Read-only.
   */
  kQTSampleTablePropertyID_TotalDecodeDuration = 'tded', /* TimeValue64, Read */
  /*
   * The least display offset in the table. (-50 is a lesser offset
   * than 20.)  Read-only.
   */
  kQTSampleTablePropertyID_MinDisplayOffset = '<ddd', /* TimeValue64, Read */
  /*
   * The greatest display offset in the table. (20 is a greater offset
   * than -50.)  Read-only.
   */
  kQTSampleTablePropertyID_MaxDisplayOffset = '>ddd', /* TimeValue64, Read */
  /*
   * 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_MinRelativeDisplayTime = '<dis', /* TimeValue64, Read
*/
  /*
   * The greatest 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' /* TimeValue64, Read */
};
```

Declared In
`Movies.h`

QTSampleTableGetNextAttributeChange Values

Constants passed to QTSampleTableGetNextAttributeChange.

```
enum {
  /*
   * Set this flag to find first num such that samples num-1 and num
   * are not adjacent, ie, dataOffset of num-1 + dataSize of num-1 !=
   * dataOffset of num
   */
  kQTSampleTableAttribute_DiscontiguousData = 1L << 0,
  /*
   * Set this flag to find the first sample with data size per sample
   * different from that of the starting sample.
   */
  kQTSampleTableAttribute_DataSizePerSampleChange = 1L << 1,
  /*
   * Set this flag to find the first sample with decode duration
   * different from that of the starting sample.
   */
  kQTSampleTableAttribute_DecodeDurationChange = 1L << 2,
  /*
   * Set this flag to find the first sample with display offset
   * different from that of the starting sample.
   */
  kQTSampleTableAttribute_DisplayOffsetChange = 1L << 3,
  /*
   * Set this flag to find the first sample with sample description ID
   * different from that of the starting sample.
   */
  kQTSampleTableAttribute_SampleDescriptionIDChange = 1L << 4,
  /*
   * Set this flag to find the first sample with any media sample flags
   * different from those of the starting sample.
   */
  kQTSampleTableAttribute_SampleFlagsChange = 1L << 5,
  /*
   * If no flags are set, find the first sample with any attribute
   * different from the starting sample.
   */
  kQTSampleTableAnyAttributeChange = 0
};
```

Declared In
`Movies.h`

QTSampleTableGetSampleFlags Values

Constants passed to QTSampleTableGetSampleFlags.

```
enum {
  mediaSampleNotSync            = 1 << 0, /* sample is not a sync sample (eg. is
frame differenced */
  mediaSampleShadowSync         = 1 << 1, /* sample is a shadow sync */
  mediaSampleDroppable          = 1 << 27, /* sample is not required to be decoded
 for later samples to be decoded properly */
  mediaSamplePartialSync        = 1 << 16, /* sample is a partial sync (e.g., I
frame after open GOP) */
  mediaSampleHasRedundantCoding = 1 << 24, /* sample is known to contain redundant
 coding */
  mediaSampleHasNoRedundantCoding = 1 << 25, /* sample is known not to contain
redundant coding */
  mediaSampleIsDependedOnByOthers = 1 << 26, /* one or more other samples depend
upon the decode of this sample */
 mediaSampleIsNotDependedOnByOthers = 1 << 27, /* synonym for mediaSampleDroppable
 */
  mediaSampleDependsOnOthers    = 1 << 28, /* sample's decode depends upon decode
 of other samples */
  mediaSampleDoesNotDependOnOthers = 1 << 29, /* sample's decode does not depend
upon decode of other samples */
  mediaSampleEarlierDisplayTimesAllowed = 1 << 30 /* samples later in decode order
 may have earlier display times */
};
```

Constants

`mediaSampleNotSync`

Returned for frame-differenced video sample data.

Available in Mac OS X v10.0 and later.

Declared in `Movies.h`.

Declared In

`Movies.h`

Movie Toolkit Reference

Framework:	Frameworks/QuickTime.framework
Declared in	Movies.h

Overview

The QuickTime Movie Toolkit helps your application construct movies, including determining what types of media to present, where movie data are located, when and how to present each data sample, and how to layer, arrange, and composite multiple movie elements.

Functions by Task

Associating Movies With Controllers

`DisposeMovieController` (page 1324)
: Disposes of a movie controller.

`NewMovieController` (page 1392)
: Locates a movie controller component and assigns a movie to that controller.

Audio Conversion and Extraction

`MovieAudioExtractionBegin` (page 1383)
: Begins a movie audio extraction session.

`MovieAudioExtractionEnd` (page 1384)
: Ends a movie audio extraction session.

`MovieAudioExtractionFillBuffer` (page 1384)
: Extracts audio from a movie.

`MovieAudioExtractionGetProperty` (page 1385)
: Gets a property of a movie audio extraction session.

`MovieAudioExtractionGetPropertyInfo` (page 1386)
: Gets information about a property of a movie audio extraction session.

`MovieAudioExtractionSetProperty` (page 1387)
: Sets a property of a movie audio extraction session.

Copying Existing Atoms

`QTCopyAtom` (page 1421)
Copies an atom and its children to a new atom container.

`QTInsertChildren` (page 1447)
Inserts a container of atoms as children of the specified parent atom.

`QTReplaceAtom` (page 1463)
Replaces the contents of an atom and its children with a different atom and its children.

`QTSwapAtoms` (page 1469)
Swaps the contents of two atoms in an atom container.

Creating and Disposing of Atom Containers

`QTDisposeAtomContainer` (page 1427)
Disposes of an atom container.

`QTNewAtomContainer` (page 1451)
Creates a new atom container.

Creating and Manipulating Sprites

`DisposeSprite` (page 1331)
Disposes of a sprite.

`GetSpriteProperty` (page 1370)
Retrieves the value of a specified sprite property.

`InvalidateSprite` (page 1379)
Invalidates the portion of a sprite's sprite world that is occupied by a sprite.

`NewSprite` (page 1410)
Creates a new sprite in a specified sprite world.

`SetSpriteProperty` (page 1491)
Sets the specified property of a sprite.

`SpriteHitTest` (page 1500)
Determines whether a location in a sprite's display coordinate system intersects the sprite.

Creating New Atoms

`QTInsertChild` (page 1446)
Creates a new child atom of the specified parent atom.

Enhancing Movie Playback Performance

`GetTrackLoadSettings` (page 1372)
Retrieves a track's preload information.

`SetMediaPlayHints` (page 1475)
Provides information to the Movie Toolbox that can influence playback of a single media.

`SetMoviePlayHints` (page 1485)
Provides information to the Movie Toolbox that can influence movie playback.

`SetTrackLoadSettings` (page 1497)
Specifies a portion of a track that is to be loaded into memory whenever it is played.

Error Functions

`QTAddMovieError` (page 1420)
Adds orthogonal errors to a movie's list of errors.

Finding and Adding Samples

`GetMediaNextInterestingDecodeTime` (page 1344)
Searches for decode times of interest in a media.

`GetMediaNextInterestingDisplayTime` (page 1345)
Searches for display times of interest in a media.

Finding Interesting Times

`GetMediaNextInterestingTime` (page 1346)
Searches for times of interest in a media.

`GetMovieNextInterestingTime` (page 1359)
Searches for times of interest in a movie's enabled tracks.

`GetTrackNextInterestingTime` (page 1373)
Searches for times of interest in a track.

High-Level Download Control

`GetMaxLoadedTimeInMovie` (page 1342)
When a movie is being progressively downloaded, returns the duration of the part of a movie that has already been downloaded.

`QTMovieNeedsTimeTable` (page 1450)
Returns whether a movie is being progressively downloaded.

High-Level Effects Functions

`QTCreateStandardParameterDialog` (page 1424)
Creates a dialog box that allows the user to choose an effect from the list of effects passed to the function.

`QTDismissStandardParameterDialog` (page 1426)
Closes a standard parameter dialog box that was created using QTCreateStandardParameterDialog.

`QTGetEffectsList` (page 1441)
Returns a QT atom container holding a list of the currently installed effects components.

`QTGetEffectsListExtended` (page 1442)
Provides for more advanced filtering of effects to be placed into the effect list.

`QTGetEffectSpeed` (page 1443)
Returns the speed of the effect, expressed in frames per second.

`QTIsStandardParameterDialogEvent` (page 1448)
Determines if a Macintosh event is processed by a standard parameter dialog box created by QTCreateStandardParameterDialog.

`QTStandardParameterDialogDoAction` (page 1467)
Lets you change some of the default behaviors of the standard parameter dialog box.

High-Level Movie Editing Functions

`NewMovieFromScrap` (page 1402)
Creates a movie from the contents of the scrap.

`PutMovieOnScrap` (page 1418)
Places a movie into the Macintosh scrap.

Low-Level Download Control

`MakeMediaTimeTable` (page 1380)
Returns a time table for the specified media.

`MakeTrackTimeTable` (page 1382)
Returns a time table for a specified track in a movie.

Metering Sound Level and Frequency

`GetMovieAudioFrequencyLevels` (page 1351)
Returns the current frequency meter levels of a movie mix.

`GetMovieAudioFrequencyMeteringBandFrequencies` (page 1351)
Returns the chosen middle frequency for each band in the configured frequency metering of a particular movie mix.

`GetMovieAudioFrequencyMeteringNumBands` (page 1352)
Returns the number of frequency bands being metered for a movie's specified audio mix.

`GetMovieAudioVolumeMeteringEnabled` (page 1355)
Returns the enabled or disabled status of volume metering of a particular audio mix of a movie.

`SetMovieAudioFrequencyMeteringNumBands` (page 1479)
Configures frequency metering for a particular audio mix in a movie.

`SetMovieAudioVolumeMeteringEnabled` (page 1481)
Enables or disables volume metering of a particular audio mix of a movie.

Modifying Atoms

`QTSetAtomData` (page 1465)
Changes the data of a leaf atom.

`QTSetAtomID` (page 1467)
Changes the ID of an atom.

Movie Functions

`CloseMovieFile` (page 1311)
Closes an open movie file.

`CreateMovieFile` (page 1316)
Creates a movie file, creates an empty movie which references the file, and opens the movie file with write permission.

`DeleteMovieFile` (page 1320)
Deletes a movie file.

`NewMovieForDataRefFromHandle` (page 1394)
Creates a movie from a public movie handle, converting internal references to external references.

`NewMovieFromDataRef` (page 1397)
Creates a movie from any device with a corresponding data handler.

`NewMovieFromFile` (page 1398)
Creates a new movie in memory from a movie file or from any type of file for which QuickTime provides an import component (AIFF, JPEG, MPEG-4, etc).

`NewMovieFromHandle` (page 1400)
Creates a movie in memory from a movie resource or a handle you obtained from PutMovieIntoHandle.

`NewMovieFromUserProc` (page 1404)
Creates a movie from data that you provide.

`OpenMovieFile` (page 1416)
Opens a specified movie file.

Movie Posters and Movie Previews

`GetPosterBox` (page 1367)
Obtains a poster's boundary rectangle.

`SetPosterBox` (page 1490)
Sets a poster's boundary rectangle.

Movies and Your Event Loop

`DisposeQTNextTaskNeededSoonerCallbackUPP` (page 1331)
Disposes of a QTNextTaskNeededSoonerCallbackUPP pointer.

`GetMovieStatus` (page 1363)
Searches for errors in all the enabled tracks of the movie and returns information about errors that are encountered during the processing associated with the MoviesTask function.

`GetTrackStatus` (page 1375)
Returns the value of the last error the media encountered while playing a specified track.

`NewQTNextTaskNeededSoonerCallbackUPP` (page 1409)
Allocates a Universal Procedure Pointer for the QTNextTaskNeededSoonerCallbackProc callback.

Registering and Unregistering Access Keys

`QTRegisterAccessKey` (page 1461)
Registers an access key.

`QTUnregisterAccessKey` (page 1470)
Removes a previously registered access key.

Removing Atoms From an Atom Container

`QTRemoveAtom` (page 1461)
Removes an atom and its children from the specified atom container.

`QTRemoveChildren` (page 1462)
Removes all the children of an atom from the specified atom container.

Retrieving Access Keys

`QTGetAccessKeys` (page 1432)
Returns all the application and system access keys of a specified access key type.

Retrieving Atoms and Atom Data

`QTCopyAtomDataToHandle` (page 1421)
Copies the specified leaf atom's data to a handle.

`QTCopyAtomDataToPtr` (page 1422)
Copies the specified leaf atom's data to a buffer.

`QTCountChildrenOfType` (page 1424)
Returns the number of atoms of a given type in the child list of the specified parent atom.

`QTFindChildByID` (page 1430)
Retrieves an atom by ID from the child list of the specified parent atom.

`QTFindChildByIndex` (page 1431)
Retrieves an atom by index from the child list of the specified parent atom.

`QTGetAtomDataPtr` (page 1433)
Retrieves a pointer to the atom data for a specified leaf atom.

`QTGetAtomTypeAndID` (page 1435)
Retrieves an atom's type and ID.

`QTGetNextChildType` (page 1445)
Returns the next atom type in the child list of the specified parent atom.

QTLockContainer (page 1449)
Locks an atom container in memory.

QTNextChildAnyType (page 1460)
Returns the next atom in the child list of the specified parent atom.

QTUnlockContainer (page 1469)
Unlocks an atom container in memory.

Saving Movies

AddMovieResource (page 1299)
Adds a movie resource to a specified resource file.

AddMovieToStorage (page 1301)
Adds a movie to a storage container that was created by CreateMovieStorage.

ClearMovieChanged (page 1310)
Sets the movie changed flag to indicate that the movie has not been changed.

CloseMovieStorage (page 1312)
Closes an open movie storage container.

CreateMovieStorage (page 1318)
Creates an empty storage location to hold a movie and opens a data handler to the stored movie with write permission.

DeleteMovieStorage (page 1321)
Deletes a movie storage container.

FlattenMovie (page 1336)
Creates a new movie file containing a specified movie.

FlattenMovieData (page 1338)
Creates a new movie and a file that contains all the movie data.

FlattenMovieDataToDataRef (page 1340)
Performs a flattening operation to a movie at a storage location.

HasMovieChanged (page 1378)
Determines whether a movie has changed and needs to be saved.

NewMovieFromDataFork (page 1395)
Retrieves a movie that is stored anywhere in the data fork of a specified Macintosh file.

NewMovieFromStorageOffset (page 1402)
Creates a new movie based on the offset to data in a storage container.

RemoveMovieResource (page 1471)
Removes a movie resource from a specified movie file.

UpdateMovieInStorage (page 1503)
Updates a movie at a storage location.

UpdateMovieResource (page 1503)
Replaces the contents of a movie resource in a specified movie file.

Setting Sound Parameters

GetMovieAudioBalance (page 1350)
Returns the balance value for the audio mix of a movie currently playing.

GetMovieAudioGain (page 1353)
Returns the gain value for the audio mix of a movie currently playing.

GetTrackAudioGain (page 1370)
Returns the gain value for the audio mix of a track currently playing.

GetTrackAudioMute (page 1371)
Returns the mute value for the audio mix of a track currently playing.

SetMovieAudioBalance (page 1478)
Sets the balance level for the mixed audio output of a movie.

SetMovieAudioGain (page 1480)
Sets the audio gain level for the mixed audio output of a movie, altering the perceived volume of the movie's playback.

SetMovieAudioMute (page 1480)
Sets the mute value for the audio mix of a movie currently playing.

SetTrackAudioGain (page 1496)
Sets the audio gain level for the audio output of a track, altering the perceived volume of the track's playback.

SetTrackAudioMute (page 1496)
Mutes or unmutes the audio output of a track.

Tween Component Requirements

QTDoTweenPtr (page 1429)
Runs a tween component and returns values in a pointer rather than a handle.

Using the Full Screen

BeginFullScreen (page 1305)
Begins full-screen mode for a specified graphics device.

EndFullScreen (page 1335)
Ends full-screen mode for a graphics device.

Working With Alternate Tracks

SetMovieLanguage (page 1484)
Specifies a movie's localized language or region code.

Working With Data References

`AddMediaDataRef` (page 1298)
: Adds a data reference to a media.

`GetMediaDataRef` (page 1342)
: Returns a copy of a specified data reference.

`GetMediaDataRefCount` (page 1344)
: Determines the number of data references in a media.

Working With Media Handler Properties

`GetMediaPropertyAtom` (page 1348)
: Retrieves the property atom container of a media handler.

`SetMediaPropertyAtom` (page 1476)
: Sets the property atom container of a media handler.

Working With Movie Restrictions

`QTCreateUUID` (page 1426)
: Creates a 128-bit universal unique ID number.

`QTEqualUUIDs` (page 1430)
: Compares two 128-bit ID numbers.

`QTGetMovieRestrictions` (page 1444)
: Returns the restrictions, if any, for a given movie.

`QTGetSupportedRestrictions` (page 1445)
: Reports the movie restrictions enforced by the currently running version of QuickTime.

`QTRestrictionsGetIndClass` (page 1463)
: Reports the class of a movie restriction.

`QTRestrictionsGetInfo` (page 1464)
: Reports information about the restrictions in a specified restriction set.

`QTRestrictionsGetItem` (page 1465)
: Retrieves specific movie restrictions.

Working With Movie Spatial Characteristics

`GetMovieColorTable` (page 1356)
: Retrieves a movie's color table.

`GetMovieSegmentDisplayBoundsRgn` (page 1362)
: Determines a movie's display boundary region for a specified segment.

`GetTrackSegmentDisplayBoundsRgn` (page 1374)
: Determines the region a track occupies in a movie's graphics world during a specified segment.

`SetMovieColorTable` (page 1482)
: Associates a ColorTable structure with a movie.

Working With Progress and Cover Functions

`GetMovieCoverProcs` (page 1356)
  Retrieves the cover functions that you set with the SetMovieCoverProcs function.

`SetMovieCoverProcs` (page 1482)
  Sets the callbacks invoked when a movie is covered or uncovered.

`SetMovieProgressProc` (page 1486)
  Attaches a progress function to a movie.

Working With Sprite Worlds

`DisposeAllSprites` (page 1322)
  Disposes of all sprites associated with a sprite world.

`DisposeSpriteWorld` (page 1332)
  Disposes of a sprite world.

`InvalidateSpriteWorld` (page 1380)
  Invalidates a rectangular area of a sprite world.

`NewSpriteWorld` (page 1411)
  Creates a new sprite world.

`SetSpriteWorldClip` (page 1493)
  Sets a sprite world's clip shape to the specified region.

`SetSpriteWorldMatrix` (page 1495)
  Sets a sprite world's matrix to the specified matrix.

`SpriteWorldHitTest` (page 1501)
  Determines whether any sprites are at a specified location in a sprite world.

`SpriteWorldIdle` (page 1502)
  Allows a sprite world to update its invalid areas.

Working With User Data

`AddUserData` (page 1302)
  Adds an item to a user data list.

`AddUserDataText` (page 1303)
  Places language-tagged text into an item in a user data list.

`CopyMediaUserData` (page 1312)
  Copies a source media's user data into a destination media's user data.

`CopyMovieUserData` (page 1313)
  Copies a source movie's user data into a destination movie's user data.

`CopyTrackUserData` (page 1314)
  Copies a source track's user data into a destination track's user data.

`CopyUserData` (page 1314)
  Copies metadata items from the source user data container to the destination user data container.

CountUserDataType (page 1315)
Determines the number of items of a given type in a user data list.

DisposeUserData (page 1335)
Disposes of a user data structure created by NewUserData.

GetNextUserDataType (page 1366)
Retrieves the next user data type in a specified user data list.

GetUserData (page 1375)
Returns a specified user data item.

GetUserDataItem (page 1376)
Returns a specified user data item.

GetUserDataText (page 1377)
Retrieves language-tagged text from an item in a user data list.

NewUserData (page 1415)
Creates a new user data structure.

NewUserDataFromHandle (page 1415)
Creates a new user data structure from a handle.

PutUserDataIntoHandle (page 1419)
Returns a handle to a user data structure.

RemoveUserData (page 1472)
Removes an item from a user data list.

RemoveUserDataText (page 1473)
Removes language-tagged text from an item in a user data list.

SetUserDataItem (page 1498)
Sets an item in a user data list.

Supporting Functions

AddMovieExecuteWiredActionsProc (page 1298)
Lets you add a callback to a movie to execute wired actions.

AddSoundDescriptionExtension (page 1302)
Adds an extension to a SoundDescription structure.

AttachMovieToCurrentThread (page 1304)
Attaches a movie to the current thread.

CanQuickTimeOpenDataRef (page 1307)
Determines whether referenced data can be opened using a graphics importer or opened in place as a movie.

CanQuickTimeOpenFile (page 1309)
Determines whether a file can be opened using a graphics importer or opened in place as a movie.

CreateShortcutMovieFile (page 1319)
Creates a movie file that just contains a reference to another movie.

DetachMovieFromCurrentThread (page 1321)
Detaches a movie from the current thread.

DisposeActionsUPP (page 1322)
Disposes of an ActionsUPP pointer.

`DisposeDoMCActionUPP` (page 1323)
: Disposes of a DoMCActionUPP pointer.

`DisposeGetMovieUPP` (page 1323)
: Disposes of a GetMovieUPP pointer.

`DisposeMovieDrawingCompleteUPP` (page 1326)
: Disposes of a MovieDrawingCompleteUPP pointer.

`DisposeMovieExecuteWiredActionsUPP` (page 1327)
: Disposes of a MovieExecuteWiredActionsUPP pointer.

`DisposeMoviePrePrerollCompleteUPP` (page 1327)
: Disposes of a MoviePrePrerollCompleteUPP pointer.

`DisposeMoviePreviewCallOutUPP` (page 1328)
: Disposes of a MoviePreviewCallOutUPP pointer.

`DisposeMovieProgressUPP` (page 1328)
: Disposes of a MovieProgressUPP pointer.

`DisposeMovieRgnCoverUPP` (page 1329)
: Disposes of a MovieRgnCoverUPP pointer.

`DisposeMoviesErrorUPP` (page 1329)
: Disposes of a MoviesErrorUPP pointer.

`DisposeQTCallBackUPP` (page 1330)
: Disposes of a QTCallBackUPP pointer.

`DisposeQTEffectListFilterUPP` (page 1330)
: Disposes of a QTEffectListFilterUPP pointer.

`DisposeQTSyncTaskUPP` (page 1331)
: Disposes of a QTSyncTaskUPP pointer.

`DisposeTextMediaUPP` (page 1333)
: Disposes of a TextMediaUPP pointer.

`DisposeTrackTransferUPP` (page 1334)
: Disposes of a TrackTransferUPP pointer.

`DisposeTweenerDataUPP` (page 1334)
: Disposes of a TweenerDataUPP pointer.

`GetMediaPlayHints` (page 1348)
: Undocumented

`GetMovieAnchorDataRef` (page 1349)
: Retrieves a movie's anchor data reference and type.

`GetMovieAudioMute` (page 1353)
: Returns the mute value for the audio mix of a movie currently playing.

`GetMovieAudioVolumeLevels` (page 1354)
: Returns the current volume meter levels of a movie.

`GetMovieDefaultDataRef` (page 1357)
: Gets a movie's default data reference.

`GetMovieLoadState` (page 1358)
: Returns a value that indicates the state of a movie's loading process.

`GetMovieProgressProc` (page 1361)
: Gets the MovieProgressProc callback attached to a movie.

GetMoviePropertyAtom (page 1361)
Gets a movie's property atom.

GetMovieThreadAttachState (page 1363)
Determines whether a given movie is attached to a thread.

GetMovieVisualBrightness (page 1364)
Returns the brightness adjustment for the movie.

GetMovieVisualContrast (page 1364)
Returns the contrast adjustment for the movie.

GetMovieVisualHue (page 1365)
Returns the hue adjustment for the movie.

GetMovieVisualSaturation (page 1366)
Returns the color saturation adjustment for the movie.

GetQuickTimePreference (page 1368)
Retrieves a particular preference from the QuickTime preferences.

GetSoundDescriptionExtension (page 1369)
Gets the current extension to a SoundDescription structure.

MovieExecuteWiredActions (page 1388)
Undocumented

MovieSearchText (page 1389)
Searches for text in a movie.

NewActionsUPP (page 1390)
Allocates a Universal Procedure Pointer for ActionsProc.

NewDoMCActionUPP (page 1391)
Allocates a Universal Procedure Pointer for the DoMCActionProc callback.

NewGetMovieUPP (page 1391)
Allocates a Universal Procedure Pointer for the GetMovieProc callback.

NewMovieDrawingCompleteUPP (page 1393)
Allocates a Universal Procedure Pointer for the MovieDrawingCompleteProc callback.

NewMovieExecuteWiredActionsUPP (page 1393)
Allocates a Universal Procedure Pointer for the MovieExecuteWiredActionsProc callback.

NewMovieFromDataFork64 (page 1396)
Provides a 64-bit version of NewMovieFromDataFork.

NewMoviePrePrerollCompleteUPP (page 1405)
Allocates a Universal Procedure Pointer for the MoviePrePrerollCompleteProc callback.

NewMoviePreviewCallOutUPP (page 1406)
Allocates a Universal Procedure Pointer for the MoviePreviewCallOutProc callback.

NewMovieProgressUPP (page 1406)
Allocates a Universal Procedure Pointer for the MovieProgressProc callback.

NewMovieRgnCoverUPP (page 1407)
Allocates a Universal Procedure Pointer for the MovieRgnCoverProc callback.

NewMoviesErrorUPP (page 1408)
Allocates a Universal Procedure Pointer for the MoviesErrorProc callback.

NewQTCallBackUPP (page 1408)
Allocates a Universal Procedure Pointer for the QTCallBackProc callback.

NewQTEffectListFilterUPP (page 1409)
Allocates a Universal Procedure Pointer for the QTEffectListFilterProc callback.

NewQTSyncTaskUPP (page 1410)
Allocates a Universal Procedure Pointer for the QTSyncTaskProc callback.

NewTextMediaUPP (page 1413)
Allocates a Universal Procedure Pointer for the TextMediaProc callback.

NewTrackTransferUPP (page 1414)
Allocates a Universal Procedure Pointer for the TrackTransferProc callback.

NewTweenerDataUPP (page 1414)
Allocates a Universal Procedure Pointer for the TweenerDataProc callback.

OpenMovieStorage (page 1417)
Opens a data handler for movie storage.

QTDisposeTween (page 1428)
Disposes of a tween component instance.

QTDoTween (page 1428)
Runs a tween component.

QTGetAtomParent (page 1434)
Gets the parent of a QT atom.

QTGetDataHandlerDirectoryDataReference (page 1436)
Returns a new data reference to the parent directory of the storage location associated with a data handler instance.

QTGetDataHandlerFullPathCFString (page 1436)
Returns the full pathname of the storage location associated with a data handler.

QTGetDataHandlerTargetNameCFString (page 1437)
Returns the name of the storage location associated with a data handler.

QTGetDataReferenceDirectoryDataReference (page 1438)
Returns a new data reference for a parent directory.

QTGetDataReferenceFullPathCFString (page 1438)
Returns the full pathname of the target of the data reference as a CFString.

QTGetDataReferenceTargetNameCFString (page 1439)
Returns the name of the target of a data reference as a CFString.

QTGetDataRefMaxFileOffset (page 1440)
Undocumented

QTNewAlias (page 1451)
Creates a Mac OS alias to a file.

QTNewDataReferenceFromCFURL (page 1452)
Creates a URL data reference from a CFURL.

QTNewDataReferenceFromFSRef (page 1453)
Creates an alias data reference from a file specification.

QTNewDataReferenceFromFSRefCFString (page 1454)
Creates an alias data reference from a file reference pointing to a directory and a file name.

QTNewDataReferenceFromFSSpec (page 1455)
Creates an alias data reference from a file specification of type FSSpec.

`QTNewDataReferenceFromFullPathCFString` (page 1456)
: Creates an alias data reference from a CFString that represents the full pathname of a file.

`QTNewDataReferenceFromURLCFString` (page 1457)
: Creates a URL data reference from a CFString that represents a URL string.

`QTNewDataReferenceWithDirectoryCFString` (page 1458)
: Creates an alias data reference from another alias data reference pointing to the parent directory and a CFString that contains the file name.

`QTNewTween` (page 1459)
: Undocumented

`RemoveMovieExecuteWiredActionsProc` (page 1471)
: Removes a MovieExecuteWiredActionsProc callback from a movie.

`RemoveSoundDescriptionExtension` (page 1472)
: Removes an extension from a SoundDescription structure.

`SetMediaDataRef` (page 1474)
: Changes the file that the specified media identifies as the location for its data storage.

`SetMediaDataRefAttributes` (page 1475)
: Sets a data reference's attributes.

`SetMovieAnchorDataRef` (page 1478)
: Sets a movie's anchor data reference and type.

`SetMovieDefaultDataRef` (page 1484)
: Sets a movie's default data reference and type.

`SetMoviePropertyAtom` (page 1487)
: Sets a movie's property atom.

`SetMovieVisualBrightness` (page 1487)
: Sets the brightness adjustment for the movie.

`SetMovieVisualContrast` (page 1488)
: Sets the contrast adjustment for the movie.

`SetMovieVisualHue` (page 1489)
: Sets the hue adjustment for the movie.

`SetMovieVisualSaturation` (page 1489)
: Sets the color saturation adjustment for the movie.

`SetQuickTimePreference` (page 1491)
: Sets a particular preference in the QuickTime preferences.

`SetSpriteWorldFlags` (page 1494)
: Sets flags that govern the behavior of a sprite world.

`SetSpriteWorldGraphicsMode` (page 1494)
: Sets the graphics transfer mode for a sprite world.

`ShowMovieInformation` (page 1499)
: Displays a movie's information.

Functions

AddMediaDataRef

Adds a data reference to a media.

```
OSErr AddMediaDataRef (
   Media theMedia,
   short *index,
   Handle dataRef,
   OSType dataRefType
);
```

Parameters

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (page 1630) and `GetTrackMedia` (page 1612). See `Media Identifiers`.

index

A pointer to a short integer. The Movie Toolbox returns the index value that is assigned to the new data reference. Your application can use this index to identify the reference to other Movie Toolbox functions, such as `GetMediaDataRef` (page 1342). If the Movie Toolbox cannot add the data reference to the media, it sets the returned index value to 0.

dataRef

The data reference. This parameter contains a handle to the information that identifies the file that contains this media's data. The type of information stored in that handle depends upon the value of the `dataRefType` parameter.

dataRefType

The type of data reference. If the data reference is an alias, you must set this parameter to `rAliasType`.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

SlideShowImporter

SlideShowImporter.win

Declared In

`Movies.h`

AddMovieExecuteWiredActionsProc

Lets you add a callback to a movie to execute wired actions.

```
OSErr AddMovieExecuteWiredActionsProc (
   Movie theMovie,
   MovieExecuteWiredActionsUPP proc,
   void *refCon
);
```

Parameters

theMovie

A movie identifier. Your application obtains this identifier from such functions as NewMovie (page 259), NewMovieFromFile (page 1398), and NewMovieFromHandle (page 1400).

proc

A callback function, as described in MovieExecuteWiredActionsProc.

refCon

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

Return Value

You can access Movie Toolbox error returns through GetMoviesError (page 221) and GetMoviesStickyError (page 222), as well as in the function result. See Error Codes.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

Movies.h

AddMovieResource

Adds a movie resource to a specified resource file.

```
OSErr AddMovieResource (
   Movie theMovie,
   short resRefNum,
   short *resId,
   ConstStr255Param resName
);
```

Parameters

theMovie

The movie you wish to add to the movie file. Your application obtains this movie identifier from such functions as NewMovie (page 259), NewMovieFromFile (page 1398), and NewMovieFromHandle (page 1400).

resRefNum

Identifies the movie file to which the resource is to be added. Your application obtains this value from the OpenMovieFile (page 1416) function.

resId

A pointer to a field that contains the resource ID number for the new resource. If the field referred to by resId is set to 0, the Movie Toolbox assigns a unique resource ID number to the new resource. The toolbox then returns the movie's resource ID number in the field referred to by the `resId` parameter. `AddMovieResource` assigns resource ID numbers sequentially, starting at 128. If resId is set to `NIL`, the Movie Toolbox assigns a unique resource ID number to the new resource and does not return that resource's ID value. Set resId to `movieInDataForkResID` to add the new resource to the movie file's data fork (see below). See these constants:

`movieInDataForkResID`

resName

Points to a character string that contains the name of the movie resource. If you set `resName` to `NIL`, the toolbox creates an unnamed resource.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

This function adds the movie to the file, effectively saving any changes you have made to the movie. To use this function with single-fork movie files, pass `movieInDataForkResID` as the `resId` parameter. After updating the movie file, `AddMovieResource` clears the movie changed flag, indicating that the movie has not been changed.

```
// AddMovieResource coding example
// See "Discovering QuickTime," page 243
void CreateMyCoolMovie (void)
{
    StandardFileReply   sfr;
    Movie               movie =NIL;
    FSSpec              fss;
    short               nFileRefNum =0;
    short               nResID =movieInDataForkResID;
    StandardPutFile("\pEnter movie file name:", "\puntitled.mov", &sfr);
    if (!sfr.sfGood)
        return;
    CreateMovieFile(&sfr.sfFile,
                    FOUR_CHAR_CODE('TVOD'),
                    smCurrentScript,
                    createMovieFileDeleteCurFile |
                     createMovieFileDontCreateResFile,
                    &nFileRefNum,
                    &movie);
    CreateMyVideoTrack(movie);      // See next section
    CreateMySoundTrack(movie);      // See next section
    AddMovieResource(movie, nFileRefNum, &nResID, NIL);
    if (nFileRefNum !=0)
        CloseMovieFile(nFileRefNum);
    DisposeMovie(movie);
}
```

Version Notes

Introduced in QuickTime 3 or earlier. Superseded in QuickTime 6 by `AddMovieToStorage` (page 1301).

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qteffects

qteffects.win

qtwiredactions

vrmakepano

vrmakepano.win

Declared In

`Movies.h`

AddMovieToStorage

Adds a movie to a storage container that was created by CreateMovieStorage.

```
OSErr AddMovieToStorage (
   Movie theMovie,
   DataHandler dh
);
```

Parameters

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

dh

The data handler component that was returned by `CreateMovieStorage` (page 1318).

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

This function calls `PutMovieIntoStorage` (page 268) internally. If you are writing a custom data handler, make sure it implements `DataHGetDataRef` (page 780). Also implement `DataHScheduleData64` (page 807) and `DataHGetFileSize64` (page 785), or `DataHScheduleData` (page 806) and `DataHGetFileSize` (page 784) if the data handler does not support 64-bit file offsets, plus `DataHWrite64` (page 819), or `DataHWrite` (page 817) if it does not support 64-bit offsets.

Version Notes

Introduced in QuickTime 6. Supersedes `AddMovieResource` (page 1299).

Availability

Available in Mac OS X v10.2 and later.

Related Sample Code

CaptureAndCompressIPBMovie

OpenGLCaptureToMovie

QTExtractAndConvertToMovieFile

Quartz Composer QCTV

SCAudioCompress

Declared In

`Movies.h`

AddSoundDescriptionExtension

Adds an extension to a SoundDescription structure.

```
OSErr AddSoundDescriptionExtension (
   SoundDescriptionHandle desc,
   Handle extension,
   OSType idType
);
```

Parameters

desc

A handle to the `SoundDescription` structure to add the extension to.

extension

The handle containing the extension data.

idType

A four-byte signature identifying the type of data being added to the `SoundDescription`.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

Two extensions are defined to the `SoundDescription` record. The first is the slope, intercept, `minClip`, and `maxClip` parameters for audio, represented as an atom of type `'flap'`. The second extension is the ability to store data specific to a given audio codec, using a `SoundDescriptionV1` structure.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

audioconverter

audioconverter.win

ConvertMovieSndTrack

soundconverter

soundconverter.win

Declared In

`Movies.h`

AddUserData

Adds an item to a user data list.

```
OSErr AddUserData (
   UserData theUserData,
   Handle data,
   OSType udType
);
```

Parameters

theUserData
: The user data list for this operation. You obtain this item reference by calling GetMovieUserData (page 225), GetTrackUserData (page 1617), or GetMediaUserData (page 1595).

data
: A handle to the data to be added to the user data list.

udType
: The type that is to be assigned to the new item.

Return Value
You can access Movie Toolbox error returns through GetMoviesError (page 221) and GetMoviesStickyError (page 222), as well as in the function result. See Error Codes.

Discussion
You specify the user data list, the data to be added, and the data's type value.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
AlwaysPreview

qtactiontargets

qtactiontargets.win

Declared In
Movies.h

AddUserDataText

Places language-tagged text into an item in a user data list.

```
OSErr AddUserDataText (
   UserData theUserData,
   Handle data,
   OSType udType,
   long index,
   short itlRegionTag
);
```

Parameters

theUserData
: The user data list for this operation. You obtain this list reference by calling GetMovieUserData (page 225), GetTrackUserData (page 1617), or GetMediaUserData (page 1595).

data

A handle to the data to be added to the user data list.

udType

The type that is to be assigned to the new item.

index

The item to which the text is to be added. This parameter must specify an item in the user data list identified by `theUserData`.

itlRegionTag

The region code of the text to be added. If there is already text with this region code in the item, the function replaces the existing text with the data specified by the `data` parameter. See *Inside Macintosh: Text* for more information about language and region codes.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

You specify the user data list and item, the data to be added, the data's type value, and the language code of the data.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

MakeEffectMovie

qtinfo

qtinfo.win

qttimecode

qttimecode.win

Declared In

`Movies.h`

AttachMovieToCurrentThread

Attaches a movie to the current thread.

```
OSErr AttachMovieToCurrentThread (
   Movie m
);
```

Parameters

m

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

Return Value

See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 6.4.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
ExtractMovieAudioToAIFF

QTAudioExtractionPanel

QTExtractAndConvertToAIFF

QTExtractAndConvertToMovieFile

Declared In
`Movies.h`

BeginFullScreen

Begins full-screen mode for a specified graphics device.

```
OSErr BeginFullScreen (
   Ptr *restoreState,
   GDHandle whichGD,
   short *desiredWidth,
   short *desiredHeight,
   WindowRef *newWindow,
   RGBColor *eraseColor,
   long flags
);
```

Parameters

restoreState

On exit, a pointer to a block of private state data that contains information on how to return from full-screen mode. This value is passed to `EndFullScreen` (page 1335) to enable it to return the monitor to its previous state.

whichGD

A handle to the graphics device to put into full-screen mode. Set this parameter to `NIL` to select the main screen.

desiredWidth

On entry, a pointer to a short integer that contains the desired width, in pixels, of the images to be displayed. On exit, that short integer is set to the actual number of pixels that can be displayed horizontally. Set this parameter to 0 to leave the width of the display unchanged.

desiredHeight

On entry, a pointer to a short integer that contains the desired height, in pixels, of the images to be displayed. On exit, that short integer is set to the actual number of pixels that can be displayed vertically. Set this parameter to 0 to leave the height of the display unchanged.

newWindow

On entry, a window-creation value. If this parameter is `NIL`, no window is created for you. If this parameter has any other value, `BeginFullScreen` creates a new window that is large enough to fill the entire screen and returns a pointer to that window in this parameter. You should not dispose of that window yourself; instead, `EndFullScreen` (page 1335) will do so.

eraseColor
The color to use when erasing the full-screen window created by `BeginFullScreen` if `newWindow` is not `NIL` on entry. If this parameter is `NIL`, `BeginFullScreen` uses black when initially erasing the window's content area.

flags
A set of bit flags (see below) that control certain aspects of the full-screen mode. See these constants:
`fullScreenHideCursor`
`fullScreenAllowEvents`
`fullScreenDontChangeMenuBar`
`fullScreenPreflightSize`

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
This function returns, in the `restoreState` parameter, a pointer to a block of private state information that indicates how to return from full-screen mode. You pass that pointer as a parameter to the `EndFullScreen` (page 1335) function. The following sample code contains functions that illustrate how to play a QuickTime movie full screen. It prompts the user for a movie, opens that movie, configures it to play full screen, associates a movie controller, and lets the controller handle events. Your application would call `QTFullScreen_EventLoopAction` in its event loop (on the Mac OS) or when it gets idle events (on Windows).

```
enum {
    fullScreenHideCursor            =1L << 0,
    fullScreenAllowEvents           =1L << 1,
    fullScreenDontChangeMenuBar     =1L << 2,
    fullScreenPreflightSize         =1L << 3
};
// QTFullScreen_PlayOnFullScreen
// Prompt the user for a movie and play it full screen.
OSErr QTFullScreen_PlayOnFullScreen (void)
{
    FSSpec              myFSSpec;
    Movie               myMovie =NIL;
    short               myRefNum =0;
    SFTypeList          myTypeList ={MovieFileType, 0, 0, 0};
    StandardFileReply   myReply;
    long                myFlags =fullScreenDontChangeMenuBar
                                    | fullScreenAllowEvents;
    OSErr               myErr =noErr;

    StandardGetFilePreview(NIL, 1, myTypeList, &myReply);
    if (!myReply.sfGood)
        goto bail;

    // make an FSSpec record
    FSMakeFSSpec(myReply.sfFile.vRefNum, myReply.sfFile.parID,
                                    myReply.sfFile.name, &myFSSpec);
    myErr =OpenMovieFile(&myFSSpec, &myRefNum, fsRdPerm);
    if (myErr !=noErr)
        goto bail;
    // now fetch the first movie from the file
    myErr =NewMovieFromFile(&myMovie, myRefNum, NIL, NIL,
                                        newMovieActive, NIL);
    if (myErr !=noErr)
        goto bail;

    CloseMovieFile(myRefNum);
    // set up for full-screen display
    myErr =BeginFullScreen(&gRestoreState, NIL, 0, 0,
                                   &gFullScreenWindow, NIL, myFlags);
#if TARGET_OS_WIN32
    // on Windows, set a window procedure for the new window
    // and associate a port with that window
    QTMLSetWindowWndProc(gFullScreenWindow, QTFullScreen_HandleMessages);
    CreatePortAssociation(GetPortNativeWindow(gFullScreenWindow), NIL, 0L);
#endif
    SetMovieGWorld(myMovie, (CGrafPtr)gFullScreenWindow,
                            GetGWorldDevice((CGrafPtr)gFullScreenWindow));
    SetMovieBox(myMovie, &gFullScreenWindow->
portRect);
    // create the movie controller
    gMC =NewMovieController(myMovie, &gFullScreenWindow->
portRect, 0);
```

Version Notes
The Macintosh human interface guidelines suggest that the menu bar must always be present, and that information must always appear in windows. However, many multimedia applications have chosen to change the look and feel of the interface based on their needs. The number of details to keep track of when doing this continues to increase. To help solve this problem, QuickTime 2.1 added functions to put a graphics device into full screen mode. The key elements to displaying full screen movies are the calls `BeginFullScreen` and `EndFullScreen`, introduced in QuickTime 2.5.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qtbigscreen

qtbigscreen.win

QTCarbonShell

qtfullscreen

qtfullscreen.win

Declared In
`Movies.h`

CanQuickTimeOpenDataRef

Determines whether referenced data can be opened using a graphics importer or opened in place as a movie.

```
OSErr CanQuickTimeOpenDataRef (
   Handle dataRef,
   OSType dataRefType,
   Boolean *outCanOpenWithGraphicsImporter,
   Boolean *outCanOpenAsMovie,
   Boolean *outPreferGraphicsImporter,
   UInt32 inFlags
);
```

Parameters

dataRef

A handle to the referenced data.

dataRefType

The type of data reference pointed to by `dataRef`; see `Data References`.

outCanOpenWithGraphicsImporter

Points to a Boolean that will be set to TRUE if the file can be opened using a graphics importer and FALSE otherwise. If you do not want this information, pass `NIL`.

outCanOpenAsMovie

Points to a Boolean that will be set to TRUE if the file can be opened as a movie and FALSE otherwise. If you do not want this information, pass `NIL`.

outPreferGraphicsImporter

Points to a boolean which will be set to true if the file can be opened using a graphics importer and opened as a movie, but, other factors being equal, QuickTime prefers a graphics importer. For example, QuickTime recommends using a graphics importer for single-frame GIF files and opening as a movie for multiple-frame GIF files. If you do not want this information, pass `NIL`. Passing a valid pointer disables the `kQTDontUseDataToFindImporter` and `kQTDontLookForMovieImporterIfGraphicsImporterFound` flags, if set.

inFlags

Flags (see below) that modify search behavior. Pass 0 for default behavior. See these constants:

`kQTDontUseDataToFindImporter`

`kQTDontLookForMovieImporterIfGraphicsImporterFound`

`kQTAllowOpeningStillImagesAsMovies`

`kQTAllowImportersThatWouldCreateNewFile`

`kQTAllowAggressiveImporters`

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

This function determines whether QuickTime can open a given area of data. You should pass `NIL` in parameters that do not interest you, since that will allow QuickTime to perform a faster determination.

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

QTCarbonCoreImage101

QTCarbonShell

Declared In

`Movies.h`

CanQuickTimeOpenFile

Determines whether a file can be opened using a graphics importer or opened in place as a movie.

```
OSErr CanQuickTimeOpenFile (
   FSSpecPtr fileSpec,
   OSType fileType,
   OSType fileNameExtension,
   Boolean *outCanOpenWithGraphicsImporter,
   Boolean *outCanOpenAsMovie,
   Boolean *outPreferGraphicsImporter,
   UInt32 inFlags
);
```

Parameters

fileSpec

Points to an `FSSpec` structure that identifies a file. To ask about a particular file type or file name suffix in general, pass `NIL`.

fileType

Contains the file type if already known, or 0 if not known. If `fileSpec` is provided and `fileType` is 0, QuickTime will determine the file type. If you pass `NIL` in `fileSpec` and 0 in `fileNameExtension`, you must pass a file type here.

fileNameExtension

Contains the file name suffix if already known, or 0 if not known. The file name suffix should be encoded as an uppercase four character code with trailing spaces; for instance, the suffix ".`png`" should be encoded as `'PNG '`, or 0x504E4720. If `fileSpec` is provided and `fileNameExtension` is 0, QuickTime will examine `fileSpec` to determine the file name suffix. If you pass `NIL` in `fileSpec` and 0 in `fileType`, you must pass a file name suffix here.

outCanOpenWithGraphicsImporter

Points to a Boolean that will be set to TRUE if the file can be opened using a graphics importer and FALSE otherwise. If you do not want this information, pass `NIL`.

outCanOpenAsMovie

Points to a Boolean that will be set to TRUE if the file can be opened as a movie and FALSE otherwise. If you do not want this information, pass `NIL`.

outPreferGraphicsImporter

Points to a boolean which will be set to true if the file can be opened using a graphics importer and opened as a movie, but, other factors being equal, QuickTime prefers a graphics importer. For example, QuickTime recommends using a graphics importer for single-frame GIF files and opening as a movie for multiple-frame GIF files. If you do not want this information, pass `NIL`. Passing a valid pointer disables the `kQTDontUseDataToFindImporter` and `kQTDontLookForMovieImporterIfGraphicsImporterFound` flags, if set.

inFlags

Flags (see below) that modify search behavior. Pass 0 for default behavior. See these constants:

`kQTDontUseDataToFindImporter`

`kQTDontLookForMovieImporterIfGraphicsImporterFound`

`kQTAllowOpeningStillImagesAsMovies`

`kQTAllowImportersThatWouldCreateNewFile`

`kQTAllowAggressiveImporters`

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

This function determines whether QuickTime can open a given file or, in general, files of a given type. You should pass `NIL` in parameters that do not interest you, since that will allow QuickTime to perform a faster determination.

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

QuickTimeMovieControl

SetCustomIcon

SimpleVideoOut

Declared In

`Movies.h`

ClearMovieChanged

Sets the movie changed flag to indicate that the movie has not been changed.

```
void ClearMovieChanged (
   Movie theMovie
);
```

Parameters

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

Return Value

You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

CloseMovieFile

Closes an open movie file.

```
OSErr CloseMovieFile (
   short resRefNum
);
```

Parameters

resRefNum

The movie file to close. Your application obtains this reference number from `OpenMovieFile` (page 1416).

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

The following code shows a typical use of `CloseMovieFile`.

```
// CloseMovieFile coding example
// See "Discovering QuickTime," page 50
void OpenMovie (HWND hwnd, char *szFileName)
{
    short   nFileRefNum =0;
    FSSpec  fss;
    // Convert path to FSSpec
    NativePathNameToFSSpec(szFileName, &fss, 0);
    // Set graphics port
    SetGWorld((CGrafPtr)GetNativeWindowPort(hwnd), NIL);
    OpenMovieFile(&fss, &nFileRefNum, fsRdPerm);   // Open movie file
    NewMovieFromFile(&movie, nFileRefNum, NIL,    // Get movie from file
                        NIL, newMovieActive, NIL);
    CloseMovieFile(nFileRefNum);                   // Close movie file
}
```

Version Notes

Introduced in QuickTime 3 or earlier. Superseded in QuickTime 6 by `CloseMovieStorage` (page 1312).

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

MakeEffectMovie

vrmakepano

VRMakePano Library

vrmakepano.win

vrscript.win

Declared In

`Movies.h`

CloseMovieStorage

Closes an open movie storage container.

```
OSErr CloseMovieStorage (
   DataHandler dh
);
```

Parameters

dh

The data handler component that was returned by a previous call to `CreateMovieStorage` (page 1318).

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 6. Supersedes `CloseMovieFile` (page 1311).

Availability

Available in Mac OS X v10.2 and later.

Related Sample Code

CaptureAndCompressIPBMovie

QTCarbonShell

QTExtractAndConvertToMovieFile

Quartz Composer QCTV

SCAudioCompress

Declared In

`Movies.h`

CopyMediaUserData

Copies a source media's user data into a destination media's user data.

```
OSErr CopyMediaUserData (
   Media srcMedia,
   Media dstMedia,
   OSType copyRule
);
```

Parameters

srcMedia

The source media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (page 1630) and `GetTrackMedia` (page 1612).

dstMedia

The destination media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (page 1630) and `GetTrackMedia` (page 1612).

`copyRule`
A constant (see below) that defines how the copying is done. See these constants:
`kQTCopyUserDataReplace`
`kQTCopyUserDataMerge`

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
Using this function is equivalent to making the following call:

```
CopyUserData(GetMediaUserData(srcMedia), GetMediaUserData(dstMedia),
             copyRule);
```

Version Notes
Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`Movies.h`

CopyMovieUserData

Copies a source movie's user data into a destination movie's user data.

```
OSErr CopyMovieUserData (
   Movie srcMovie,
   Movie dstMovie,
   OSType copyRule
);
```

Parameters

`srcMovie`
The source movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

`dstMovie`
The destination movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

`copyRule`
A constant (see below) that defines how the copying is done. See these constants:
`kQTCopyUserDataReplace`
`kQTCopyUserDataMerge`

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
Using this function is equivalent to making the following call:

```
CopyUserData(GetMovieUserData(srcMovie), GetMovieUserData(dstMovie),
             copyRule);
```

Version Notes
Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`Movies.h`

CopyTrackUserData

Copies a source track's user data into a destination track's user data.

```
OSErr CopyTrackUserData (
   Track srcTrack,
   Track dstTrack,
   OSType copyRule
);
```

Parameters

srcTrack
The source track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

dstTrack
The destination track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

copyRule
A constant (see below) that defines how the copying is done. See these constants:
`kQTCopyUserDataReplace`
`kQTCopyUserDataMerge`

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`Movies.h`

CopyUserData

Copies metadata items from the source user data container to the destination user data container.

```
OSErr CopyUserData (
   UserData srcUserData,
   UserData dstUserData,
   OSType copyRule
);
```

Parameters

srcUserData

The source user data list for this operation. You obtain this list reference by calling GetMovieUserData (page 225), GetTrackUserData (page 1617), or GetMediaUserData (page 1595).

dstUserData

The destination user data list for this operation. You obtain this list reference by calling GetMovieUserData (page 225), GetTrackUserData (page 1617), or GetMediaUserData (page 1595).

copyRule

A constant (see below) that defines how the copying is done. See these constants:

`kQTCopyUserDataReplace`

`kQTCopyUserDataMerge`

Return Value

You can access Movie Toolbox error returns through GetMoviesError (page 221) and GetMoviesStickyError (page 222), as well as in the function result. See `Error Codes`.

Discussion

The function detects if the source and destination containers already have the same content and does nothing in that case.

Version Notes

Introduced in QuickTime 6.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`Movies.h`

CountUserDataType

Determines the number of items of a given type in a user data list.

```
short CountUserDataType (
   UserData theUserData,
   OSType udType
);
```

Parameters

theUserData

The user data list for this operation. You obtain this list reference by calling the GetMovieUserData (page 225), GetTrackUserData (page 1617), or GetMediaUserData (page 1595) functions.

udType

The type. The Movie Toolbox determines the number of items of this type in the user data list.

Return Value

The number of items of the given type in the user data list.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Graphic Import-Export

MakeEffectMovie

qtactiontargets

qtactiontargets.win

qteffects.win

Declared In

`Movies.h`

CreateMovieFile

Creates a movie file, creates an empty movie which references the file, and opens the movie file with write permission.

```
OSErr CreateMovieFile (
   const FSSpec *fileSpec,
   OSType creator,
   ScriptCode scriptTag,
   long createMovieFileFlags,
   short *resRefNum,
   Movie *newmovie
);
```

Parameters

fileSpec

A pointer to the file system specification for the movie file to be created.

creator

The creator value for the new file.

scriptTag

The script in which the movie file should be created. Use the Script Manager constant `smSystemScript` to use the system script; use the `smCurrentScript` constant to use the current script. See *Inside Macintosh: Text* for more information about scripts and script tags.

createMovieFileFlags

Controls movie file creation flags (see below). See these constants:

`createMovieFileDontCreateResFile`

`createMovieFileDeleteCurFile`

`createMovieFileDontCreateMovie`

`createMovieFileDontOpenFile`

`newMovieActive`

resRefNum

A pointer to a field that is to receive the file reference number for the opened movie file. Your application must use this value when calling other Movie Toolbox functions that work with movie files. If you set this parameter to `NIL`, the Movie Toolbox creates the movie file but does not open the file.

newmovie

A pointer to a field that is to receive the identifier of the new movie. `CreateMovieFile` returns the identifier of the new movie. If the function could not create a new movie, it sets this returned value to `NIL`. If you set this parameter to `NIL`, the Movie Toolbox does not create a movie.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

The following code snippet shows how `CreateMovieFile` may be used to create and open a QuickTime movie file.

```
// CreateMovieFile coding example
// See "Discovering QuickTime," page 243
void CreateMyCoolMovie (void)
{
    StandardFileReply   sfr;
    Movie               movie =NIL;
    FSSpec              fss;
    short               nFileRefNum =0;
    short               nResID =movieInDataForkResID;
    StandardPutFile("\pEnter movie file name:", "\puntitled.mov", &sfr);
    if (!sfr.sfGood)
        return;
    CreateMovieFile(&sfr.sfFile,
                    FOUR_CHAR_CODE('TVOD'),
                    smCurrentScript,
                    createMovieFileDeleteCurFile |
                     createMovieFileDontCreateResFile,
                    &nFileRefNum,
                    &movie);
    CreateMyVideoTrack(movie);   // See "Discovering QuickTime," page 244
    CreateMySoundTrack(movie);   // See "Discovering QuickTime," page 250
    AddMovieResource(movie, nFileRefNum, &nResID, NIL);
    if (nFileRefNum !=0)
        CloseMovieFile(nFileRefNum);
    DisposeMovie(movie);
}
```

Version Notes

Introduced in QuickTime 3 or earlier. Superseded in QuickTime 6 by `CreateMovieStorage` (page 1318).

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qteffects

qteffects.win

vrmakepano

VRMakePano Library

vrmakepano.win

Declared In
`Movies.h`

CreateMovieStorage

Creates an empty storage location to hold a movie and opens a data handler to the stored movie with write permission.

```
OSErr CreateMovieStorage (
   Handle dataRef,
   OSType dataRefType,
   OSType creator,
   ScriptCode scriptTag,
   long createMovieFileFlags,
   DataHandler *outDataHandler,
   Movie *newmovie
);
```

Parameters

dataRef
A handle to a QuickTime data reference.

dataRefType
The data reference type. See `Data References`.

creator
The creator type of the new container (for example, 'TV0D', the `creator` type for Apple's movie player).

scriptTag
Constants (see below) that specify the script for the new container. See these constants:

createMovieFileFlags
Constants (see below) that control file creation options. See these constants:
`createMovieFileDeleteCurFile`
`createMovieFileDontCreateMovie`
`createMovieFileDontOpenFile`
`newMovieActive`

outDataHandler
A pointer to a field that is to receive the data handler for the opened movie container. Your application must use this value when calling other Movie Toolbox functions. If you pass `NIL`, the Movie Toolbox creates the movie container but does not open it.

newmovie
A pointer to a field that is to receive the returned identifier of the new movie. If the function could not create a new movie, it sets this returned value to `NIL`. If you pass `NIL`, the Movie Toolbox does not create a movie.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
If you are writing a custom data handler, make sure it supports `DataHGetDataRef` (page 780). It must also support `DataHWrite64` (page 819), or `DataHWrite` (page 817) if 64-bit offsets are not supported.

Version Notes
Introduced in QuickTime 6. Supersedes `CreateMovieFile` (page 1316).

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
CaptureAndCompressIPBMovie

OpenGLCaptureToMovie

QTKitCreateMovie

Quartz Composer QCTV

SCAudioCompress

Declared In
`Movies.h`

CreateShortcutMovieFile

Creates a movie file that just contains a reference to another movie.

```
OSErr CreateShortcutMovieFile (
   const FSSpec *fileSpec,
   OSType creator,
   ScriptCode scriptTag,
   long createMovieFileFlags,
   Handle targetDataRef,
   OSType targetDataRefType
);
```

Parameters

`fileSpec`
: A pointer to the file system specification for the movie file to be created.

`creator`
: The creator value for the new file.

`scriptTag`
: The script in which the movie file should be created. Use the Script Manager constant `smSystemScript` to use the system script; use the `smCurrentScript` constant to use the current script. See *Inside Macintosh: Text* for more information about scripts and script tags.

`createMovieFileFlags`
: Contains movie file creation flags (see below). See these constants:

  `flattenAddMovieToDataFork`

  `flattenDontInterleaveFlatten`

  `flattenActiveTracksOnly`

  `flattenCompressMovieResource`

  `flattenFSSpecPtrIsDataRefRecordPtr`

  `flattenForceMovieResourceBeforeMovieData` *targetDataRef*

A handle to the data referred to by the movie that this function creates.

targetDataRefType

The type of the data referred to by the movie that this function creates; see `Data References`.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qtshortcut

qtshortcut.win

Declared In

`Movies.h`

DeleteMovieFile

Deletes a movie file.

```
OSErr DeleteMovieFile (
   const FSSpec *fileSpec
);
```

Parameters

fileSpec

A pointer to the file system specification for the movie file to be deleted.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier. Superseded in QuickTime 6 by `DeleteMovieStorage` (page 1321).

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qtstreamsplicer.win

Sequence Grabbing

vrmakepano

VRMakePano Library

vrmakepano.win

Declared In

`Movies.h`

DeleteMovieStorage

Deletes a movie storage container.

```
OSErr DeleteMovieStorage (
   Handle dataRef,
   OSType dataRefType
);
```

Parameters

dataRef

A handle to a QuickTime data reference that identifies the movie storage to be deleted.

dataRefType

The data reference type. See `Data References`.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

If you are writing a custom data handler that supports this call, make sure that it implements `DataHDeleteFile` (page 773).

Version Notes

Introduced in QuickTime 6. Supersedes `DeleteMovieFile` (page 1320).

Availability

Available in Mac OS X v10.2 and later.

Declared In

`Movies.h`

DetachMovieFromCurrentThread

Detaches a movie from the current thread.

```
OSErr DetachMovieFromCurrentThread (
   Movie m
);
```

Parameters

m

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

Return Value

See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 6.4.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

ExtractMovieAudioToAIFF

QTAudioExtractionPanel

QTExtractAndConvertToAIFF

QTExtractAndConvertToMovieFile

Declared In
`Movies.h`

DisposeActionsUPP

Disposes of an ActionsUPP pointer.

```
void DisposeActionsUPP (
   ActionsUPP userUPP
);
```

Parameters

userUPP
An `ActionsUPP` pointer. See `Universal Procedure Pointers`.

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

DisposeAllSprites

Disposes of all sprites associated with a sprite world.

```
void DisposeAllSprites (
   SpriteWorld theSpriteWorld
);
```

Parameters

theSpriteWorld
The sprite world for this operation.

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Discussion
This function calls `DisposeSprite` (page 1331) for each sprite associated with the sprite world.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

DisposeDoMCActionUPP

Disposes of a DoMCActionUPP pointer.

```
void DisposeDoMCActionUPP (
   DoMCActionUPP userUPP
);
```

Parameters

userUPP

A `DoMCActionUPP` pointer. See `Universal Procedure Pointers`.

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

DisposeGetMovieUPP

Disposes of a GetMovieUPP pointer.

```
void DisposeGetMovieUPP (
   GetMovieUPP userUPP
);
```

Parameters

userUPP

A `GetMovieUPP` pointer. See `Universal Procedure Pointers`.

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

DisposeMovieController

Disposes of a movie controller.

```
void DisposeMovieController (
   ComponentInstance mc
);
```

Parameters

mc

The movie controller for the operation. You obtain this identifier from the Component Manager's `OpenComponent` or `OpenDefaultComponent` function, or from the `NewMovieController` (page 1392) function.

Return Value

You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Discussion

This function is implemented by the Movie Toolbox, not by movie controller components. If you are creating your own movie controller component, you do not have to support this function. The following code snippet illustrates its use:

```
// DisposeMovieController coding example
// See "Discovering QuickTime," page 221
// Resource identifiers
#define IDM_OPEN         101
char            szMovieFile[MAX_PATH];              // Name of movie file
Movie           movie;                              // Movie object
MovieController mc;                                 // Movie controller
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                      LPSTR lpCmdLine, int nCmdShow)
{
    ...
    ...
    InitializeQTML(0);                          // Initialize QuickTime
    EnterMovies();                              // Initialize Toolbox
    ...
    //  Main message loop
    ...
    ExitMovies();                               // Terminate Toolbox
    TerminateQTML();                            // Terminate QuickTime
}  // end WinMain
//
LRESULT CALLBACK WndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
    MSG             msg;
    EventRecord     er;

    . . .                                       // Fill in contents of MSG
structure

    WinEventToMacEvent(&msg, &er);                  // Convert message to a QT
 event
    MCIsPlayerEvent(mc, (const EventRecord *)&er);  // Pass event to movie
controller

    switch (iMsg) {
        case WM_CREATE:
            CreatePortAssociation(hwnd, NIL, OL);  // Register window with QT
            break;
        case WM_COMMAND:
            switch (LOWORD(wParam)) {
                case IDM_OPEN:
                    MyCloseMovie();                 // Close previous movie, if
 any

                    if (MyGetFile(szMovieFile))         // Get file name from
user
                        MyOpenMovie(hwnd, szMovieFile); // Open the movie
                    break;
                    . . .
                default:
                    return DefWindowProc(hwnd, iMsg, wParam, lParam);
            }  // end switch (LOWORD(wParam))
            break;
        case WM_CLOSE:
            DestroyPortAssociation(GetNativeWindowPort(hwnd));  // Unregister
window
            break;
        . . .
        default:
            return DefWindowProc(hwnd, iMsg, wParam, lParam);

    }  // end switch (iMsg)

    return 0;
}  // end WndProc
//
BOOL MyGetFile (char *lpszMovieFile)
{
    OPENFILENAME         ofn;

    // Fill in contents of OPENFILENAME structure
                ...
                ...

     if (GetOpenFileName(&ofn))                      // Let user select file
        return TRUE;
    else
        return FALSE;
}  // end MyGetFile
//
void MyOpenMovie (HWND hwnd, char szFileName[255])
{
    short   nFileRefNum =0;
    FSSpec  fss;
    SetGWorld((CGrafPtr)GetNativeWindowPort(hwnd), NIL);   // Set graphics port
    NativePathNameToFSSpec(szFileName, &fss, 0);    // Convert pathname and make
 FSSpec
    OpenMovieFile(&fss, &nFileRefNum, fsRdPerm);    // Open movie file
    NewMovieFromFile(&movie, nFileRefNum, NIL,     // Get movie from file
                    NIL, newMovieActive, NIL);
    CloseMovieFile(nFileRefNum);                    // Close movie file

    mc =NewMovieController(movie, ...);            // Make movie controller
```

```
        ...
        ...

}  // end MyOpenMovie
//
void MyCloseMovie (void)
{
    if (mc)                                    // Destroy movie controller, if
 any
        DisposeMovieController(mc);

    if (movie)                                 // Destroy movie object, if any
        DisposeMovie(movie);
}  // end MyCloseMovie
```

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CarbonQTGraphicImport

MakeEffectMovie

qtstreamsplicer.win

vrscript

vrscript.win

Declared In

`Movies.h`

DisposeMovieDrawingCompleteUPP

Disposes of a MovieDrawingCompleteUPP pointer.

```
void DisposeMovieDrawingCompleteUPP (
   MovieDrawingCompleteUPP userUPP
);
```

Parameters

userUPP

A `MovieDrawingCompleteUPP` pointer. See `Universal Procedure Pointers`.

Return Value

You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes

Introduced in QuickTime 4.1.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

ASCIIMoviePlayerSample

ASCIIMoviePlayerSample for Windows
OpenGLMovieQT
VideoProcessing

Declared In
`Movies.h`

DisposeMovieExecuteWiredActionsUPP

Disposes of a MovieExecuteWiredActionsUPP pointer.

```
void DisposeMovieExecuteWiredActionsUPP (
   MovieExecuteWiredActionsUPP userUPP
);
```

Parameters

userUPP
A `MovieExecuteWiredActionsUPP` pointer. See `Universal Procedure Pointers`.

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

DisposeMoviePrePrerollCompleteUPP

Disposes of a MoviePrePrerollCompleteUPP pointer.

```
void DisposeMoviePrePrerollCompleteUPP (
   MoviePrePrerollCompleteUPP userUPP
);
```

Parameters

userUPP
A `MoviePrePrerollCompleteUPP` pointer. See `Universal Procedure Pointers`.

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code

vrscript

vrscript.win

Declared In

`Movies.h`

DisposeMoviePreviewCallOutUPP

Disposes of a MoviePreviewCallOutUPP pointer.

```
void DisposeMoviePreviewCallOutUPP (
   MoviePreviewCallOutUPP userUPP
);
```

Parameters

userUPP

A `MoviePreviewCallOutUPP` pointer. See `Universal Procedure Pointers`.

Return Value

You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes

Introduced in QuickTime 4.1.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

DisposeMovieProgressUPP

Disposes of a MovieProgressUPP pointer.

```
void DisposeMovieProgressUPP (
   MovieProgressUPP userUPP
);
```

Parameters

userUPP

A `MovieProgressUPP` pointer. See `Universal Procedure Pointers`.

Return Value

You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes

Introduced in QuickTime 4.1.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

BackgroundExporter

qtdataexchange

qtdataexchange.win

Declared In

`Movies.h`

DisposeMovieRgnCoverUPP

Disposes of a MovieRgnCoverUPP pointer.

```
void DisposeMovieRgnCoverUPP (
   MovieRgnCoverUPP userUPP
);
```

Parameters

`userUPP`

A `MovieRgnCoverUPP` pointer. See `Universal Procedure Pointers`.

Return Value

You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes

Introduced in QuickTime 4.1.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

DisposeMoviesErrorUPP

Disposes of a MoviesErrorUPP pointer.

```
void DisposeMoviesErrorUPP (
   MoviesErrorUPP userUPP
);
```

Parameters

`userUPP`

A `MoviesErrorUPP` pointer. See `Universal Procedure Pointers`.

Return Value

You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes

Introduced in QuickTime 4.1.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

DisposeQTCallBackUPP

Disposes of a QTCallBackUPP pointer.

```
void DisposeQTCallBackUPP (
   QTCallBackUPP userUPP
);
```

Parameters

userUPP
A `QTCallBackUPP` pointer. See `Universal Procedure Pointers`.

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qtbigscreen

qtbigscreen.win

Declared In
`Movies.h`

DisposeQTEffectListFilterUPP

Disposes of a QTEffectListFilterUPP pointer.

```
void DisposeQTEffectListFilterUPP (
   QTEffectListFilterUPP userUPP
);
```

Parameters

userUPP
A `QTEffectListFilterUPP` pointer. See `Universal Procedure Pointers`.

Version Notes
Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`Movies.h`

DisposeQTNextTaskNeededSoonerCallbackUPP

Disposes of a QTNextTaskNeededSoonerCallbackUPP pointer.

```
void DisposeQTNextTaskNeededSoonerCallbackUPP (
   QTNextTaskNeededSoonerCallbackUPP userUPP
);
```

Parameters

userUPP

A `QTNextTaskNeededSoonerCallbackUPP` pointer. See `Universal Procedure Pointers`.

Version Notes

Introduced in QuickTime 6.

Availability

Available in Mac OS X v10.2 and later.

Related Sample Code

qtshellCEvents

qtshellCEvents.win

VideoProcessing

Declared In

`Movies.h`

DisposeQTSyncTaskUPP

Disposes of a QTSyncTaskUPP pointer.

```
void DisposeQTSyncTaskUPP (
   QTSyncTaskUPP userUPP
);
```

Parameters

userUPP

A `QTSyncTaskUPP` pointer. See `Universal Procedure Pointers`.

Return Value

You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes

Introduced in QuickTime 4.1.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

DisposeSprite

Disposes of a sprite.

```
void DisposeSprite (
   Sprite theSprite
);
```

Parameters

theSprite

The sprite to be disposed of.

Return Value

You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Discussion

You call this function to dispose of a sprite created by `NewSprite` (page 1410). The image description handle and image data pointer associated with the sprite are not disposed of by this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Desktop Sprites

DesktopSprites

DesktopSprites.win

Declared In

`Movies.h`

DisposeSpriteWorld

Disposes of a sprite world.

```
void DisposeSpriteWorld (
   SpriteWorld theSpriteWorld
);
```

Parameters

theSpriteWorld

The sprite world to dispose of. It is safe to pass `NIL` to this function.

Return Value

You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Discussion

You call this function to dispose of a sprite world created by `NewSpriteWorld` (page 1411). This function also disposes of all of the sprites associated with the sprite world. This function does not dispose of the graphics worlds associated with the sprite world. Here is an example of using it:

```
// DisposeSpriteWorld coding example
// See "Discovering QuickTime," page 347
#define kNumSprites            4
#define kNumSpaceShipImages    24
SpriteWorld                 gSpriteWorld =NIL;
```

```
Sprite                      gSprites[kNumSprites];
Handle                      gCompressedPictures[kNumSpaceShipImages];
ImageDescriptionHandle      gImageDescriptions[kNumSpaceShipImages];
void MyDisposeEverything (void)
{
    short           nIndex;
    // dispose of the sprite world and associated graphics world
    if (gSpriteWorld)
        DisposeSpriteWorld(gSpriteWorld);

    // dispose of each sprite's image data
    for (nIndex =0; nIndex < kNumSprites; nIndex++) {
        if (gCompressedPictures[nIndex])
            DisposeHandle(gCompressedPictures[nIndex]);
        if (gImageDescriptions[nIndex])
            DisposeHandle((Handle)gImageDescriptions[nIndex]);
    }
    DisposeGWorld(spritePlane);
}
```

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Desktop Sprites

DesktopSprites

DesktopSprites.win

Declared In
`Movies.h`

DisposeTextMediaUPP

Disposes of a TextMediaUPP pointer.

```
void DisposeTextMediaUPP (
   TextMediaUPP userUPP
);
```

Parameters

userUPP

A `TextMediaUPP` pointer. See `Universal Procedure Pointers`.

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code

qttext

qttext.win

Declared In

`Movies.h`

DisposeTrackTransferUPP

Disposes of a TrackTransferUPP pointer.

```
void DisposeTrackTransferUPP (
   TrackTransferUPP userUPP
);
```

Parameters

`userUPP`

A `TrackTransferUPP` pointer. See `Universal Procedure Pointers`.

Return Value

You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes

Introduced in QuickTime 4.1.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

DisposeTweenerDataUPP

Disposes of a TweenerDataUPP pointer.

```
void DisposeTweenerDataUPP (
   TweenerDataUPP userUPP
);
```

Parameters

`userUPP`

A `TweenerDataUPP` pointer. See `Universal Procedure Pointers`.

Return Value

You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes

Introduced in QuickTime 4.1.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

DisposeUserData

Disposes of a user data structure created by NewUserData.

```
OSErr DisposeUserData (
   UserData theUserData
);
```

Parameters
theUserData

The user data structure that is to be disposed of. It is acceptable but unnecessary to pass `NIL` in this parameter.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Graphic Import-Export

QTKitTimeCode

qttimecode

qttimecode.win

WhackedTV

Declared In
`Movies.h`

EndFullScreen

Ends full-screen mode for a graphics device.

```
OSErr EndFullScreen (
   Ptr fullState,
   long flags
);
```

Parameters
fullState

The pointer to private state information returned by a previous call to `BeginFullScreen` (page 1305).

flags

Reserved. Set this parameter to `NIL`.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
This function restores the graphics device and other settings to the state specified by the private state information pointed to by the `fullState` parameter. The resulting state is that that was in effect prior to the immediately previous call to `BeginFullScreen` (page 1305). The following code illustrates its use:

```
OSErr QTFullScreen_RestoreScreen (void)
{
    OSErr       myErr =noErr;

#if TARGET_OS_WIN32
    DestroyPortAssociation((CGrafPtr)gFullScreenWindow);
#endif
    DisposeMovieController(gMC);
    myErr =EndFullScreen(gRestoreState, 0L);

    return(myErr);
}
```

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
FullScreen

qtbigscreen

QTCarbonShell

qtfullscreen

qtfullscreen.win

Declared In
`Movies.h`

FlattenMovie

Creates a new movie file containing a specified movie.

```
void FlattenMovie (
   Movie theMovie,
   long movieFlattenFlags,
   const FSSpec *theFile,
   OSType creator,
   ScriptCode scriptTag,
   long createMovieFileFlags,
   short *resId,
   ConstStr255Param resName
);
```

Parameters

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

movieFlattenFlags

Contains flags (see below) that control the process of adding movie data to the new movie file. Set unused flags to 0. See these constants:

`flattenAddMovieToDataFork`

`flattenDontInterleaveFlatten`

`flattenActiveTracksOnly`

`flattenCompressMovieResource`

`flattenFSSpecPtrIsDataRefRecordPtr`

`flattenForceMovieResourceBeforeMovieData`

theFile

A pointer to the file system specification for the movie file to be created.

creator

The creator value for the new file.

scriptTag

The script in which the movie file should be created. Set this parameter to the Script Manager constant `smSystemScript` to use the system script; set it to `smCurrentScript` to use the current script. See *Inside Macintosh: Text* for more information about scripts and script tags.

createMovieFileFlags

Contains flags (see below) that control file creation options. See these constants:

`createMovieFileDeleteCurFile`

resId

A pointer to a field that contains the resource ID number for the new resource. If the field referred to by the `resId` parameter is set to 0, the Movie Toolbox assigns a unique resource ID number to the new resource. The toolbox then returns the movie's resource ID number in the field referred to by the `resId` parameter. The Movie Toolbox assigns resource ID numbers sequentially, starting at 128. If the `resId` parameter is set to `NIL`, the Movie Toolbox assigns a unique resource ID number to the new resource and does not return that resource's ID value.

resName

Points to a character string with the name of the movie resource. If you set the `resName` parameter to `NIL`, the toolbox creates an unnamed resource.

Return Value

You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Discussion
The file created by `FlattenMovie` also contains all the data for the movie; that is, the Movie Toolbox resolves any data references and includes the corresponding movie data in the new movie file.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
AddFrameToMovie

mfc.win

MovieGWorlds

simpleeditsdi.win

simpleplayersdi.win

Declared In
`Movies.h`

FlattenMovieData

Creates a new movie and a file that contains all the movie data.

```
Movie FlattenMovieData (
   Movie theMovie,
   long movieFlattenFlags,
   const FSSpec *theFile,
   OSType creator,
   ScriptCode scriptTag,
   long createMovieFileFlags
);
```

Parameters

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

movieFlattenFlags

Contains flags (see below) that control the process of adding movie data to the new movie file. These flags affect how the toolbox adds movies to the new movie file later. Set unused flags to 0. See these constants:

`flattenAddMovieToDataFork`

`flattenDontInterleaveFlatten`

`flattenActiveTracksOnly`

`flattenCompressMovieResource`

`flattenFSSpecPtrIsDataRefRecordPtr`

`flattenForceMovieResourceBeforeMovieData`

theFile

This parameter usually contains a pointer to the file system specification for the movie file to be created. In place of a `FSSpec` pointer, QuickTime lets you pass a pointer to a data reference structure to receive the flattened movie data.

`creator`

The creator value for the new file.

`scriptTag`

Contains constants (see below) that specify the script in which the movie file should be created. See *Inside Macintosh: Text* for more information about scripts and script tags. See these constants:

`createMovieFileFlags`

Contains flags (see below) that control file creation options. See these constants:

`createMovieFileDeleteCurFile`

Return Value

The identifier of the new movie. If the function could not create the movie, it sets this returned identifier to `NIL`.

Discussion

This function will take any movie and optionally make it self-contained, interleaved, and Fast Start. Unlike `FlattenMovie` (page 1336), this function does not add the new movie resource to the new movie file; instead, `FlattenMovieData` returns the new movie to your application. Your application must dispose of the returned movie. You can use this function to create a single-fork movie file, by setting the `flattenAddMovieToDataFork` flag in the `movieFlattenFlags` parameter to 1. The Movie Toolbox then places the movie into the data fork of the movie file. Instead of flattening to a file, you can specify a data reference to flatten a movie to. The following two code samples show flattening a movie to a data location and to a file:

```
// FlattenMovieData used to flatten a movie to a data location
// create a 0-length handle
    myHandle =NewHandleClear(mySize);
    if (myHandle ==NIL)
        goto bail;

// fill in the data reference record
    myDataRefRec.dataRefType =HandleDataHandlerSubType;
    myDataRefRec.dataRef =NewHandle(sizeof(Handle));
    if (myDataRefRec.dataRef ==NIL)
        goto bail;
    *((Handle *)*(myDataRefRec.dataRef)) =myHandle;
    myFlags =flattenFSSpecPtrIsDataRefRecordPtr;
    myFile =(FSSpec *)&myDataRefRec;
    // flatten the source movie into the handle
    myMemMovie =FlattenMovieData(mySrcMovie, myFlags, myFile, 0L,
                              smSystemScript, 0L);
    Movie aMovie;
    aMovie =FlattenMovieData(theMovie,
        flattenAddMovieToDataFork |
        flattenForceMovieResourceBeforeMovieData,
        &theOutputFile, OSTypeConst('TVOD'), smSystemScript,
        createMovieFileDeleteCurFile | createMovieFileDontCreateResFile);

    DisposeMovie(aMovie);
    Movie aMovie;
    aMovie =FlattenMovieData(theMovie,
        flattenAddMovieToDataFork,
        &theOutputFile, OSTypeConst('TVOD'), smSystemScript,
        createMovieFileDeleteCurFile | createMovieFileDontCreateResFile);

    DisposeMovie(aMovie);
```

```
// FlattenMovieData used to flatten a movie to a Fast Start file
// See "Discovering QuickTime," page 257
myErr =OpenMovieFile(&myTempSpec, &myTempResRefNum, fsRdPerm);
if (myErr !=noErr)
    goto bail;
myErr =NewMovieFromFile(&myTempMovie, myTempResRefNum, NIL, 0, 0, 0);
if (myErr !=noErr)
    goto bail;
SetMovieProgressProc(myTempMovie, (MovieProgressUPP)-1, 0L);
// flatten the temporary file into a new movie file; put the movie
// resource first so that progressive downloading is possible
myPanoMovie =FlattenMovieData(
                        myTempMovie,
                        flattenDontInterleaveFlatten
                        | flattenAddMovieToDataFork
                        | flattenForceMovieResourceBeforeMovieData,
                        &myDestSpec,
                        FOUR_CHAR_CODE('TVOD'),
                        smSystemScript,
                        createMovieFileDeleteCurFile
                        | createMovieFileDontCreateResFile);
```

Special Considerations

Through the `SetTrackLoadSettings` (page 1497) function, the Movie Toolbox allows you to set a movie's preloading guidelines when you create the movie. The preload information is preserved when you save or flatten the movie (using either `FlattenMovie` or `FlattenMovieData`). In flattened movies, the tracks that are to be preloaded are stored at the start of the movie, rather than being interleaved with the rest of the movie data. This greatly improves preload performance because it is not necessary for the device storing the movie data to seek during retrieval of the data to be preloaded.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qtdataref

vrmakeobject

vrmakepano

VRMakePano Library

vrmakepano.win

Declared In

`Movies.h`

FlattenMovieDataToDataRef

Performs a flattening operation to a movie at a storage location.

```
Movie FlattenMovieDataToDataRef (
   Movie theMovie,
   long movieFlattenFlags,
   Handle dataRef,
   OSType dataRefType,
   OSType creator,
   ScriptCode scriptTag,
   long createMovieFileFlags
);
```

Parameters

theMovie

The movie passed into this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

movieFlattenFlags

Constants (see below) that control the process of adding movie data to the new container. QuickTime will read these flags later when it adds movies to the storage. Set unused flags to 0. See these constants:

`flattenAddMovieToDataFork`

`flattenDontInterleaveFlatten`

`flattenActiveTracksOnly`

`flattenCompressMovieResource`

`flattenForceMovieResourceBeforeMovieData`

dataRef

A handle to a QuickTime data reference.

dataRefType

The data reference type. See `Data References`.

creator

The creator type of the new container (for example, 'TV0D', the `creator` type for Apple's movie player).

scriptTag

Constants (see below) that specify the script for the new container. See these constants:

createMovieFileFlags

Constants (see below) that control file creation options. See these constants:

`createMovieFileDeleteCurFile`

`createMovieFileDontCreateMovie`

`createMovieFileDontOpenFile`

Return Value

The identifier of the new movie. If the function could not create the movie, it sets the returned identifier to `NIL`.

Discussion

This function performs a flattening operation to the destination data reference.

Version Notes

Introduced in QuickTime 6.

Availability

Available in Mac OS X v10.2 and later.

Related Sample Code
QTCarbonShell

Declared In
`Movies.h`

GetMaxLoadedTimeInMovie

When a movie is being progressively downloaded, returns the duration of the part of a movie that has already been downloaded.

```
OSErr GetMaxLoadedTimeInMovie (
   Movie theMovie,
   TimeValue *time
);
```

Parameters

`theMovie`

The movie for this operation. Your application obtains this identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

`time`

The duration of the part of a movie that has already been downloaded. This time value is expressed in the movie's time coordinate system. If all of a movie has been downloaded, this parameter returns the duration of the entire movie.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
The Movie Toolbox creates a time table for a movie when either `QTMovieNeedsTimeTable` (page 1450) or `GetMaxLoadedTimeInMovie` is called for the movie, but the time table is used only by the toolbox and is not accessible to applications. The toolbox disposes of the time table when the download is complete.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

GetMediaDataRef

Returns a copy of a specified data reference.

```
OSErr GetMediaDataRef (
   Media theMedia,
   short index,
   Handle *dataRef,
   OSType *dataRefType,
   long *dataRefAttributes
);
```

Parameters

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (page 1630) and `GetTrackMedia` (page 1612). See `Media Identifiers`.

index

The index value that corresponds to the data reference. It must be less than or equal to the value that is returned by `GetMediaDataRefCount` (page 1344).

dataRef

A pointer to a field that is to receive a handle to the data reference. The media handler returns a handle to information that identifies the file that contains this media's data. The type of information stored in that handle depends upon the value of the `dataRefType` parameter. If the function cannot locate the specified data reference, the handler sets this returned value to `NIL`. Set the `dataRef` parameter to `NIL` if you are not interested in this information.

dataRefType

A pointer to a field that is to receive the type of data reference. If the data reference is an alias, the media handler sets this value to `'alis'`. Set the `dataRefType` parameter to `NIL` if you are not interested in this information.

dataRefAttributes

A pointer to a field that is to receive the reference's attribute flags (see below). Unused flags are set to 0. See these constants:

`dataRefSelfReference`

`dataRefWasNotResolved`

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

Use this function to retrieve information about a data reference. For example, you might want to verify the condition of a movie's data references after loading the movie from its movie file. You could use this function to check each data reference.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

BurntTextSampleCode

SlideShowImporter

SlideShowImporter.win

ThreadsImporter

ThreadsImportMovie

Declared In
`Movies.h`

GetMediaDataRefCount

Determines the number of data references in a media.

```
OSErr GetMediaDataRefCount (
   Media theMedia,
   short *count
);
```

Parameters

theMedia
The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (page 1630) and `GetTrackMedia` (page 1612). See `Media Identifiers`.

count
A pointer to a field that is to receive the number of data references in the media.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
ThreadsImporter

ThreadsImportMovie

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` (page 1630) and `GetTrackMedia` (page 1612).

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. See these constants:

`nextTimeMediaSample`

`nextTimeMediaEdit`

`nextTimeSyncSample`

`nextTimeEdgeOK`

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.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
MovieVideoChart

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` (page 1630) and `GetTrackMedia` (page 1612).

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. See these constants:

`nextTimeMediaSample`

`nextTimeMediaEdit`

`nextTimeSyncSample`

`nextTimeEdgeOK`

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.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`Movies.h`

GetMediaNextInterestingTime

Searches for times of interest in a media.

```
void GetMediaNextInterestingTime (
   Media theMedia,
   short interestingTimeFlags,
   TimeValue time,
   Fixed rate,
   TimeValue *interestingTime,
   TimeValue *interestingDuration
);
```

Parameters

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (page 1630) and `GetTrackMedia` (page 1612). See `Media Identifiers`.

interestingTimeFlags

Contains flags (see below) 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. See these constants:

`nextTimeMediaSample`

`nextTimeMediaEdit`

`nextTimeSyncSample`

`nextTimeEdgeOK`

time

Specifies a time value that establishes the starting point for the search. 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.

interestingTime

A pointer to a time value. 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.

interestingDuration

A pointer to a time value. The Movie Toolbox returns the duration of the interesting time. This time value is 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.

Return Value

You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Discussion

Some compression algorithms conserve space by eliminating duplication between consecutive frames in a sample. They do this by deriving frames from sync samples, which don't rely on preceding frames for content.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code
qdmediahandler

qdmediahandler.win

TimeCode Media Handlers

Declared In
`Movies.h`

GetMediaPlayHints

Undocumented

```
void GetMediaPlayHints (
   Media theMedia,
   long *flags
);
```

Parameters

`theMedia`
: The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (page 1630) and `GetTrackMedia` (page 1612). See `Media Identifiers`.

`flags`
: *Undocumented*

Return Value
Undocumented

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

GetMediaPropertyAtom

Retrieves the property atom container of a media handler.

```
OSErr GetMediaPropertyAtom (
   Media theMedia,
   QTAtomContainer *propertyAtom
);
```

Parameters

`theMedia`
: A reference to the media handler for this operation.

`propertyAtom`
: A pointer to a QT atom container. On return, the atom container contains the property atoms for the track associated with the media handler.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

You can call `GetMediaPropertyAtom` to retrieve the properties of the track associated with the specified media handler. The contents of the returned QT atom container are defined by the media handler.

Special Considerations

The caller is responsible for disposing of the QT atom container.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

addvractions

addvractions.win

vrscript

vrscript.win

Declared In

`Movies.h`

GetMovieAnchorDataRef

Retrieves a movie's anchor data reference and type.

```
OSErr GetMovieAnchorDataRef (
   Movie theMovie,
   Handle *dataRef,
   OSType *dataRefType,
   long *outFlags
);
```

Parameters

`theMovie`

A movie identifier. Your application obtains this identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

`dataRef`

A handle to the `data` reference. The type of information stored in the handle depends upon the `data` reference type specified by `dataRefType`.

`dataRefType`

The type of data reference; see `Data References`.

`outFlags`

If there is no anchor data reference associated with the movie, then `GetMovieAnchorDataRef` sets this parameter to `kMovieAnchorDataRefIsDefault` (see below) and returns copies of the default data reference and type. See these constants:

`kMovieAnchorDataRefIsDefault`

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

If there is neither an anchor nor a default data reference, `NIL` will be returned in `dataRef` and 0 in `dataRefType`.

Special Considerations

The caller should dispose of the data reference returned.

Version Notes

Introduced in QuickTime 4.1.

Availability

Available in Mac OS X v10.0 and later.

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` (page 259), `NewMovieFromProperties` (page 260), `NewMovieFromFile`, and `NewMovieFromHandle` (page 1400).

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 1478).

Availability

Available in Mac OS X v10.3 and later.

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` (page 259), `NewMovieFromProperties` (page 260), `NewMovieFromFile`, and `NewMovieFromHandle` (page 1400).

`whatMixToMeter`

The applicable mix of audio channels in the movie; see `Movie Audio Mixes`.

`pAveragePowerLevels`

A pointer to a `QTAudioFrequencyLevels` structure (page 325).

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.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

Core Animation QuickTime Layer

SillyFrequencyLevels

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` (page 259), `NewMovieFromProperties` (page 260), `NewMovieFromFile`, and `NewMovieFromHandle` (page 1400).

whatMixToMeter
: The applicable mix of audio channels in the movie; see `Movie Audio Mixes`.

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.

Availability
Available in Mac OS X v10.3 and later.

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` (page 259), `NewMovieFromProperties` (page 260), `NewMovieFromFile`, and `NewMovieFromHandle` (page 1400).

whatMixToMeter
: The applicable mix of audio channels in the movie; see `Movie Audio Mixes`.

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 1479).

Availability
Available in Mac OS X v10.3 and later.

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` (page 259), `NewMovieFromProperties` (page 260), `NewMovieFromFile`, and `NewMovieFromHandle` (page 1400).

gain

A 32-bit floating-point gain value of 0 or greater. This value is multiplied by the movie's volume. 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 1480).

Availability
Available in Mac OS X v10.3 and later.

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` (page 259), `NewMovieFromProperties` (page 260), `NewMovieFromFile`, and `NewMovieFromHandle` (page 1400).

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 1480).

Availability

Available in Mac OS X v10.3 and later.

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` (page 259), `NewMovieFromProperties` (page 260), `NewMovieFromFile`, and `NewMovieFromHandle` (page 1400).

whatMixToMeter

The applicable mix of audio channels in the movie; see `Movie Audio Mixes`.

pAveragePowerLevels

A pointer to a `QTAudioVolumeLevels` structure that stores the average power level of each channel in the mix, measured in decibels. A return of `NIL` means no channels; if non-NIL, 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.

pPeakHoldLevels

A pointer to a QTAudioVolumeLevels structure that stores the peak hold level of each channel in the mix, measured in decibels. A return of NIL means no channels; if non-NIL, 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.

Return Value

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

Discussion

If either pAveragePowerLevels or pPeakHoldLevels returns non-NIL, 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.

Availability

Available in Mac OS X v10.3 and later.

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 (page 259), NewMovieFromProperties (page 260), NewMovieFromFile, and NewMovieFromHandle (page 1400).

whatMixToMeter

The applicable mix of audio channels in the movie; see Movie Audio Mixes.

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 1481).

Availability

Available in Mac OS X v10.3 and later.

Declared In

Movies.h

GetMovieColorTable

Retrieves a movie's color table.

```
OSErr GetMovieColorTable (
   Movie theMovie,
   CTabHandle *ctab
);
```

Parameters

theMovie

The movie for this operation. Your application obtains this identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

ctab

A pointer to a field that is to receive a handle to the movie's color table. If the movie does not have a color table, the toolbox sets the field to `NIL`.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

The toolbox returns a copy of the color table, so it is your responsibility to dispose of the color table when you are done with it.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

GetMovieCoverProcs

Retrieves the cover functions that you set with the SetMovieCoverProcs function.

```
OSErr GetMovieCoverProcs (
   Movie theMovie,
   MovieRgnCoverUPP *uncoverProc,
   MovieRgnCoverUPP *coverProc,
   long *refcon
);
```

Parameters

theMovie

The movie for this operation. Your application obtains this identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

uncoverProc

Where to return the current uncover procedure. This value is set to `NIL` if no uncover procedure was specified.

coverProc

Where to return the current cover procedure. This value is set to `NIL` if no cover procedure was specified.

refcon

A reference constant to be passed to your callback. Use this parameter to point to a data structure containing any information your cover functions need.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

This function returns the uncover and cover functions for the movie as well as the reference constant for the cover functions.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

GetMovieDefaultDataRef

Gets a movie's default data reference.

```
OSErr GetMovieDefaultDataRef (
   Movie theMovie,
   Handle *dataRef,
   OSType *dataRefType
);
```

Parameters

theMovie

A movie identifier. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

dataRef

A pointer to a field that is to receive a handle to the data reference. The function returns a handle to information that identifies the file that contains this media's data. The type of information stored in that handle depends upon the value of the `dataRefType` parameter. If the function cannot locate the specified data reference, the handler sets this returned value to `NIL`. Set the `dataRef` parameter to `NIL` if you are not interested in this information.

dataRefType

A pointer to a field that is to receive the type of data reference; see `Data References`. If the data reference is an alias, the function sets this value to `'alis'`, indicating that the reference is an alias. Set the `dataRefType` parameter to `NIL` if you are not interested in this information.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

GetMovieLoadState

Returns a value that indicates the state of a movie's loading process.

```
long GetMovieLoadState (
   Movie theMovie
);
```

Parameters

theMovie

A movie identifier. Your application obtains this identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

Return Value
A constant (see below) that indicates the movie's loading status.

Discussion
This function lets your code perform relative comparisons against movie loading milestones to determine if certain operations make sense. Its return values are ordered so that they conform to this rule:

```
kMovieLoadStateError
< kMovieLoadStateLoading
< kMovieLoadStatePlayable
< kMovieLoadStateComplete
```

Special Considerations

Because of the "voting system" involved, an application checking for the load state should throttle its calling of the routine. Not calling `GetMovieLoadState` more often than every quarter of a second is a good place to start.

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Movie From DataRef

QTCarbonShell

Declared In
`Movies.h`

GetMovieNextInterestingTime

Searches for times of interest in a movie's enabled tracks.

```
void GetMovieNextInterestingTime (
   Movie theMovie,
   short interestingTimeFlags,
   short numMediaTypes,
   const OSType *whichMediaTypes,
   TimeValue time,
   Fixed rate,
   TimeValue *interestingTime,
   TimeValue *interestingDuration
);
```

Parameters

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

interestingTimeFlags

Contains flags (see below) that determine the search criteria. Note that you may set only one of the `nextTimeMediaSample`, `nextTimeMediaEdit`, `nextTimeTrackEdit` and `nextTimeSyncSample` flags to 1. Set unused flags to 0. See these constants:

`nextTimeMediaSample`

`nextTimeMediaEdit`

`nextTimeTrackEdit`

`nextTimeSyncSample`

`nextTimeStep`

`nextTimeEdgeOK`

`nextTimeIgnoreActiveSegment`

numMediaTypes

The number of media types in the table referred to by the `whichMediaType` parameter. Set this parameter to 0 to search all media types.

whichMediaTypes

A pointer to an array of media type constants (see below). You can use this parameter to limit the search to a specified set of media types. Each entry in the table referred to by this parameter identifies a media type to be included in the search. You use the `numMediaTypes` parameter to indicate the number of entries in the `table`. Set this parameter to `NIL` to search all media types. See these constants:

`VisualMediaCharacteristic`

`AudioMediaCharacteristic`

time

Specifies a time value that establishes the starting point for the search. This time value must be expressed in the movie'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.

interestingTime

A pointer to a time value. 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 movie's time scale. If there are no times that meet the search criteria you specify, the Movie Toolbox sets this value to -1. If you are not interested in this information, set this parameter to `NIL`.

interestingDuration

A pointer to a time value. The Movie Toolbox returns the duration of the interesting time. This time value is in the movie's time coordinate system. Set this parameter to `NIL` if you don't want this information; in this case, the function works faster.

Discussion

The following code sample shows the use of `GetMovieNextInterestingTime` to return, through the `time` parameter, the starting time of the first video sample of the specified QuickTime movie. The trick here is to set the `nextTimeEdgeOK` flag, to indicate that you want to get the starting time of the beginning of the movie. If this function encounters an error, it returns a (bogus) starting time of -1, as shown below:

```
static OSErr QTStep_GetStartTimeOfFirstVideoSample (Movie theMovie,
                                                    TimeValue *theTime)
{
    short           myFlags;
    OSType          myTypes[1];

    *theTime =kBogusStartingTime;               // a bogus starting time
    if (theMovie ==NIL)
        return(invalidMovie);

    myFlags =nextTimeMediaSample + nextTimeEdgeOK;
                                  // we want the first sample in the movie
    myTypes[0] =VisualMediaCharacteristic;      // we want video samples
    GetMovieNextInterestingTime(theMovie, myFlags, 1, myTypes,
                                       (TimeValue)0, fixed1, theTime, NIL);
    return(GetMoviesError());
}
```

Special Considerations

This function examines only the movie's enabled tracks.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CompressMovies

DigitizerShell

DragAndDrop Shell

MovieGWorlds

QT Internals

Declared In

`Movies.h`

GetMovieProgressProc

Gets the MovieProgressProc callback attached to a movie.

```
void GetMovieProgressProc (
   Movie theMovie,
   MovieProgressUPP *p,
   long *refcon
);
```

Parameters

theMovie

A movie identifier. Your application obtains this identifier from such functions as NewMovie (page 259), NewMovieFromFile (page 1398), and NewMovieFromHandle (page 1400).

p

On return, a pointer to a `MovieProgressProc` callback.

refcon

On return, a reference constant passed to the `callback`. This parameter is used to point to a data structure containing any information the function needs.

Return Value

You can access this function's error returns through GetMoviesError (page 221) and GetMoviesStickyError (page 222).

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

GetMoviePropertyAtom

Gets a movie's property atom.

```
OSErr GetMoviePropertyAtom (
   Movie theMovie,
   QTAtomContainer *propertyAtom
);
```

Parameters

theMovie

A movie identifier. Your application obtains this identifier from such functions as NewMovie (page 259), NewMovieFromFile (page 1398), and NewMovieFromHandle (page 1400).

propertyAtom

A pointer to a property atom.

Return Value

You can access Movie Toolbox error returns through GetMoviesError (page 221) and GetMoviesStickyError (page 222), as well as in the function result. See `Error Codes`.

Discussion

This routine is used to author event handlers for the `kQTEventMovieLoaded` QuickTime event.

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

GetMovieSegmentDisplayBoundsRgn

Determines a movie's display boundary region for a specified segment.

```
RgnHandle GetMovieSegmentDisplayBoundsRgn (
   Movie theMovie,
   TimeValue time,
   TimeValue duration
);
```

Parameters

theMovie
: The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

time
: The starting time of the movie segment to consider. This time value must be expressed in the movie's time coordinate system. The duration parameter specifies the length of the segment.

duration
: The length of the segment to consider. Set this parameter to 0 to specify an instant in time.

Return Value
A handle to a `MacRegion` structure that the function allocates. This region is defined in the movie's display coordinate system. If the movie does not have a spatial representation at the current time, the function returns an empty region. If the function could not satisfy the request, it sets the returned handle to `NIL`.

Discussion
This function allocates a region and returns a handle to it. The Movie Toolbox derives the display boundary region only from enabled tracks and only from those tracks that are used in the current display mode (movie, poster, or preview). The display boundary region encloses all of a movie's enabled tracks after the track matrix, track clip, movie matrix, and movie clip have been applied to them.

Special Considerations

Your application must dispose of the returned region when it is done with it.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

GetMovieStatus

Searches for errors in all the enabled tracks of the movie and returns information about errors that are encountered during the processing associated with the MoviesTask function.

```
ComponentResult GetMovieStatus (
   Movie theMovie,
   Track *firstProblemTrack
);
```

Parameters

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

firstProblemTrack

A pointer to a track identifier. The Movie Toolbox places the identifier for the first track that is found to contain an error into the field referred to by this parameter. If you don't want to receive the track identifier, set this parameter to `NIL`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error in the movie status value.

Discussion

This function returns information about errors that are encountered during `MoviesTask` (page 257) execution. These errors typically reflect playback problems, such as low-memory conditions. `GetMovieStatus` returns the error associated with the first problem track.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Movie From DataRef

ThreadsImportMovie

Declared In

`Movies.h`

GetMovieThreadAttachState

Determines whether a given movie is attached to a thread.

```
OSErr GetMovieThreadAttachState (
   Movie m,
   Boolean *outAttachedToCurrentThread,
   Boolean *outAttachedToAnyThread
);
```

Parameters

m

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

outAttachedToCurrentThread

A pointer to a Boolean that on exit is TRUE if the movie is attached to the current thread, FALSE otherwise.

outAttachedToAnyThread

A pointer to a Boolean that on exit is TRUE if the movie is attached to any thread, FALSE otherwise.

Return Value
See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 6.4.

Availability
Available in Mac OS X v10.3 and later.

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.

Availability
Available in Mac OS X v10.3 and later.

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 gray.

Availability

Available in Mac OS X v10.3 and later.

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.

Availability
Available in Mac OS X v10.3 and later.

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.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`Movies.h`

GetNextUserDataType

Retrieves the next user data type in a specified user data list.

```
long GetNextUserDataType (
   UserData theUserData,
   OSType udType
);
```

Parameters

theUserData
The user data list for this operation. You obtain this list reference by calling `GetMovieUserData` (page 225), `GetTrackUserData` (page 1617), or `GetMediaUserData` (page 1595).

udType
: Specifies a user data field; see `User Data Identifiers`. Set this parameter to 0 to retrieve the first user data field in the user data list. On subsequent requests, use the previous value returned by this function.

Return Value
The next user data type in the list. Returns 0 when there are no more user data types.

Discussion
Use this function to scan all the user data types in a user data list.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Graphic Import-Export

ImproveYourImage

Declared In
`Movies.h`

GetPosterBox

Obtains a poster's boundary rectangle.

```
void GetPosterBox (
   Movie theMovie,
   Rect *boxRect
);
```

Parameters

theMovie
: The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

boxRect
: A pointer to a rectangle. The Movie Toolbox returns the poster's boundary rectangle into the structure referred to by this parameter.

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

GetQuickTimePreference

Retrieves a particular preference from the QuickTime preferences.

```
OSErr GetQuickTimePreference (
   OSType preferenceType,
   QTAtomContainer *preferenceAtom
);
```

Parameters

preferenceType

A preference type to be retrieved (see below); see `Atom ID Codes`. See these constants:

`ConnectionSpeedPrefsType`

`BandwidthManagementPrefsType`

preferenceAtom

A pointer to the returned preference atom.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

The following sample code shows how to retrieve the connection speed setting from the QuickTime preferences:

```
struct ConnectionSpeedPrefsRecord {
   long connectionSpeed;
};
typedef struct ConnectionSpeedPrefsRecord ConnectionSpeedPrefsRecord;
. . .
OSErr                       err;
QTAtomContainer             prefs;
QTAtom                      prefsAtom;
long                        dataSize;
Ptr                         atomData;
ConnectionSpeedPrefsRecord  prefrec;
err =GetQuickTimePreference(ConnectionSpeedPrefsType, &amp;prefs);
if (err ==noErr) {
    prefsAtom =QTFindChildByID(prefs, kParentAtomIsContainer,
                                      ConnectionSpeedPrefsType, 1, nil);
    if (!prefsAtom) {
        // set the default setting to 28.8kpbs
        prefrec.connectionSpeed =kDataRate288ModemRate;
    } else {
        err =QTGetAtomDataPtr(prefs, prefsAtom, &dataSize,
                                                     &atomData);
        if (dataSize !=sizeof(ConnectionSpeedPrefsRecord)) {
            // the prefs record wasn't the right size,
            // so it must be corrupt -- set to the default
            prefrec.connectionSpeed =kDataRate288ModemRate;
        } else {
            // everything was fine -- read the connection speed
            prefrec =*(ConnectionSpeedPrefsRecord *)atomData;
        }
    }
    QTDisposeAtomContainer(prefs);
}
```

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
MakeEffectMovie

qteffects.win

qtgraphics.win

qtwiredactions

vrbackbuffer.win

Declared In
`Movies.h`

GetSoundDescriptionExtension

Gets the current extension to a SoundDescription structure.

```
OSErr GetSoundDescriptionExtension (
   SoundDescriptionHandle desc,
   Handle *extension,
   OSType idType
);
```

Parameters

`desc`

A handle to a `SoundDescription` structure.

`extension`

A pointer to a handle that, on return, contains the extension.

`idType`

A four-byte signature that identifies the type of data in the extension.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
ConvertMovieSndTrack

SoundPlayer

SoundPlayer.win

Declared In
`Movies.h`

GetSpriteProperty

Retrieves the value of a specified sprite property.

```
OSErr GetSpriteProperty (
   Sprite theSprite,
   long propertyType,
   void *propertyValue
);
```

Parameters

theSprite

The sprite for this operation.

propertyType

The property whose value should be retrieved (see below). See these constants:

`kSpritePropertyMatrix`

`kSpritePropertyImageDescription`

`kSpritePropertyImageDataPtr`

`kSpritePropertyVisible`

`kSpritePropertyLayer`

`kSpritePropertyGraphicsMode`

`kSpritePropertyCanBeHitTested`

propertyValue

A pointer to a variable that will hold the selected property value on return. Depending on the `property` type, this parameter is either a pointer to the property value or the property value itself, cast as a `void` pointer.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

You call this function to retrieve the value of a sprite property, setting the `propertyType` parameter to the type of the property you want to retrieve.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

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` (page 1628) and `GetMovieTrack` (page 1601).

gain

A 32-bit floating-point gain value of 0 or greater. This value is multiplied by the track's volume. 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 1496).

Availability

Available in Mac OS X v10.3 and later.

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` (page 1628) and `GetMovieTrack` (page 1601).

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 1496).

Availability
Available in Mac OS X v10.3 and later.

Declared In
`Movies.h`

GetTrackLoadSettings

Retrieves a track's preload information.

```
void GetTrackLoadSettings (
   Track theTrack,
   TimeValue *preloadTime,
   TimeValue *preloadDuration,
   long *preloadFlags,
   long *defaultHints
);
```

Parameters

theTrack
The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

preloadTime
Specifies a field to receive the starting point of the portion of the track to be preloaded. The toolbox returns a value of -1 if the entire track is to be preloaded.

preloadDuration
Specifies a field to receive the amount of the track to be preloaded, starting from the time specified in the `preloadTime` parameter. If the entire track is to be preloaded, this value is ignored.

preloadFlags
Specifies a field to receive the flags (see below) that control when the toolbox preloads the track. See these constants:
`preloadAlways`
`preloadOnlyIfEnabled`

defaultHints
Specifies a field to receive the playback hints for the track.

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
SimpleVideoOut

Declared In
`Movies.h`

GetTrackNextInterestingTime

Searches for times of interest in a track.

```
void GetTrackNextInterestingTime (
   Track theTrack,
   short interestingTimeFlags,
   TimeValue time,
   Fixed rate,
   TimeValue *interestingTime,
   TimeValue *interestingDuration
);
```

Parameters

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

interestingTimeFlags

Contains flags (see below) that determine the search criteria. Note that you may set only one of the `nextTimeMediaSample`, `nextTimeMediaEdit`, `nextTimeTrackEdit` and `nextTimeSyncSample` flags to 1. Set unused flags to 0. See these constants:

`nextTimeMediaSample`

`nextTimeMediaEdit`

`nextTimeTrackEdit`

`nextTimeSyncSample`

`nextTimeEdgeOK`

`nextTimeIgnoreActiveSegment`

time

Specifies a time value that establishes the starting point for the search. This time value must be expressed in the movie'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.

interestingTime

A pointer to a time value. 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 movie'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.

interestingDuration

A pointer to a time value. The Movie Toolbox returns the duration of the interesting time. This time value is in the movie's time coordinate system. Set this parameter to `NIL` if you don't want this information; in this case, the function works more quickly.

Discussion
Some compression algorithms conserve space by eliminating duplication between consecutive frames in a sample. In this case, sync samples don't rely on preceding frames for content. You can access error returns from this function through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
BurntTextSampleCode

MovieVideoChart

qttext

qttext.win

qtwiredactions

Declared In
`Movies.h`

GetTrackSegmentDisplayBoundsRgn

Determines the region a track occupies in a movie's graphics world during a specified segment.

```
RgnHandle GetTrackSegmentDisplayBoundsRgn (
   Track theTrack,
   TimeValue time,
   TimeValue duration
);
```

Parameters

theTrack
: The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

time
: The starting time of the track segment to consider. This time value must be expressed in the movie's time coordinate system. The duration parameter specifies the length of the segment.

duration
: The length of the segment to consider. Set this parameter to 0 to consider an instant in time.

Return Value
A handle to the region the specified track occupies in its movie's graphics world during a specified segment. If the track does not have a spatial representation during the specified segment, the function returns an empty region. If the function could not satisfy your request, it sets the returned handle to `NIL`.

Discussion
This function allocates the region and returns a handle to it. This region is valid for the specified segment.

Special Considerations

Your application must dispose of the returned region when you are done with it.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
BurntTextSampleCode

Declared In
`Movies.h`

GetTrackStatus

Returns the value of the last error the media encountered while playing a specified track.

```
ComponentResult GetTrackStatus (
   Track theTrack
);
```

Parameters

theTrack
The track for this operation. Your application obtains this track identifier from `GetMovieStatus` (page 1363).

Return Value
`GetTrackStatus` returns the last error encountered for the specified track; see `Error Codes`. If the component does not find any errors, the result is set to `noErr`.

Discussion
This function returns information about errors that are encountered during the processing associated with `MoviesTask` (page 257). These errors typically reflect playback problems, such as low-memory conditions. This function returns the last error encountered for the specified track. The media clears this error code when it detects that the error has been corrected.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

GetUserData

Returns a specified user data item.

```
OSErr GetUserData (
   UserData theUserData,
   Handle data,
   OSType udType,
   long index
);
```

Parameters

theUserData

The user data list for this operation. You obtain this list reference by calling the `GetMovieUserData` (page 225), `GetTrackUserData` (page 1617), or `GetMediaUserData` (page 1595) function.

data

A handle that is to receive the data from the specified item. `GetUserData` resizes this handle as appropriate to accommodate the item. Your application is responsible for releasing this handle when you are done with it. Set this parameter to `NIL` if you don't want to retrieve the user data item. This can be useful if you want to verify that a user data item exists, but you don't need to work with the item's contents.

udType

The item's type value; see `User Data Identifiers`.

index

The item's index value. This parameter must specify an item in the user data list identified by the parameter `theUserData`.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Graphic Import-Export

MakeEffectMovie

qtactiontargets

qtactiontargets.win

qteffects.win

Declared In

`Movies.h`

GetUserDataItem

Returns a specified user data item.

```
OSErr GetUserDataItem (
   UserData theUserData,
   void *data,
   long size,
   OSType udType,
   long index
);
```

Parameters

theUserData
The user data list for this operation. You obtain this list reference by calling the `GetMovieUserData` (page 225), `GetTrackUserData` (page 1617), or `GetMediaUserData` (page 1595).

data
A pointer that is to receive the data from the specified item.

size
The size of the item.

udType
The item's type value; see `User Data Identifiers`.

index
The item's index value. This parameter must specify an item in the user data list identified by the parameter `theUserData`.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
MakeEffectMovie

qtcontroller

qtmusic.win

qtshellCEvents.win

samplemakeeffectmovie.win

Declared In
`Movies.h`

GetUserDataText

Retrieves language-tagged text from an item in a user data list.

```
OSErr GetUserDataText (
   UserData theUserData,
   Handle data,
   OSType udType,
   long index,
   short itlRegionTag
);
```

Parameters

theUserData

The user data list for this operation. You obtain this list reference by calling the GetMovieUserData (page 225), GetTrackUserData (page 1617), or GetMediaUserData (page 1595) function.

data

A handle that is to receive the data. The GetUserDataText function resizes this handle as appropriate. Your application must dispose of the handle when you are done with it.

udType

The item's type value; see User Data Identifiers.

index

The item's index value. This parameter must specify an item in the user data list identified by the parameter theUserData.

itlRegionTag

The language code of the text to be retrieved. See Localization Codes.

Return Value

You can access Movie Toolbox error returns through GetMoviesError (page 221) and GetMoviesStickyError (page 222), as well as in the function result. See Error Codes.

Discussion

You specify the user data list and item, and the item's type value and language code. The Movie Toolbox retrieves the specified text from the user data item.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Graphic Import-Export

QTCarbonShell

qtinfo

QTKitTimeCode

qttimecode.win

Declared In

Movies.h

HasMovieChanged

Determines whether a movie has changed and needs to be saved.

```
Boolean HasMovieChanged (
   Movie theMovie
);
```

Parameters

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

Return Value

Returns TRUE if the movie has changed, FALSE otherwise.

Discussion

Your application can clear the movie changed flag, indicating that the movie has not changed, by calling `ClearMovieChanged` (page 1310).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

InvalidateSprite

Invalidates the portion of a sprite's sprite world that is occupied by a sprite.

```
void InvalidateSprite (
   Sprite theSprite
);
```

Parameters

theSprite

The sprite for this operation.

Return Value

You can access error returns from this function through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). See `Error Codes`.

Discussion

In most cases, you do not need to call this function. When you call `SetSpriteProperty` (page 1491) to modify a sprite's properties, it takes care of invalidating the appropriate regions of the sprite world. However, you might call this function if you change a sprite's image data but retain the same image data pointer.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

InvalidateSpriteWorld

Invalidates a rectangular area of a sprite world.

```
OSErr InvalidateSpriteWorld (
   SpriteWorld theSpriteWorld,
   Rect *invalidArea
);
```

Parameters

theSpriteWorld
The sprite world for this operation.

invalidArea
A pointer to the `Rect` structure that defines the area that should be invalidated. This rectangle should be specified in the sprite world's source space, which is the coordinate system of the sprite layer's graphics world before the sprite world's matrix is applied to it. To invalidate the entire sprite world, pass `NIL` for this parameter.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

Typically, your application calls this function when the sprite world's destination window receives an update event. Invalidating an area of the sprite world will cause the area to be redrawn the next time that `SpriteWorldIdle` (page 1502) is called.

Special Considerations

When you modify sprite properties, invalidation takes place automatically; you do not need to call this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

MakeMediaTimeTable

Returns a time table for the specified media.

```
ComponentResult ADD_MEDIA_BASENAME() MakeMediaTimeTable
```

Parameters

theMedia
The media for this operation. Your application obtains this identifier from such functions as `NewTrackMedia` (page 1630) and `GetTrackMedia` (page 1612).

offsets
A handle to an unlocked relocatable memory block allocated by your application. The function returns the time table for the media in this block.

startTime
: The first point of the media to be included in the time table. This time value is expressed in the media's time coordinate system.

endTime
: The last point of the media to be included in the time table. This time value is expressed in the media's time coordinate system.

timeIncrement
: The resolution of the time table. The values in a time table are for a points in the media, and these points are separated by the amount of time specified by this parameter. The time value is expressed in the media's time coordinate system.

firstDataRefIndex
: An index to the first data reference for the media to be included in the time table. Set this parameter to -1 to include all data references for the `media`. Set this parameter to 1 to specify the first data reference for the media.

lastDataRefIndex
: An index to the last data reference for the media to be included in the time table. The value 1 specifies the first data reference for the media. If the value of the `firstDataRefIndex` parameter is -1, set this parameter to 0.

retdataRefSkew
: The offset to the next row of the time table, in long integers. The next row contains values for the next data reference, as explained below. By adding the `value` of this parameter to an offset into the table, you get the offset to the corresponding point for the next data reference.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
Your application must allocate an unlocked relocatable memory block for the time table to be returned and pass a handle to it in the `offsets` parameter. The `MakeMediaTimeTable` (page 1380) function resizes the block to accommodate the time table it returns.

This time table is a two-dimensional array of long integers, organized so that each row in the table contains values for one data reference. The first column in the table contains values for the time in the media specified by the `startTime` parameter, and each subsequent column contains values for the point in the media that is later by the value specified by the `timeIncrement` parameter. Each long integer value in the table specifies the offset, in bytes, from the beginning of the data reference for that point in the media. The number of columns in the table is equal to `(endTime - startTime) / timeIncrement`, rounded up. Because of alignment issues, this value is not always the same as the value of the `retdataRefSkew` parameter.

Special Considerations

When all the data for a movie has been transferred, your application must dispose of the time table created by this function.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

MakeTrackTimeTable

Returns a time table for a specified track in a movie.

```
OSErr MakeTrackTimeTable (
   Track trackH,
   long **offsets,
   TimeValue startTime,
   TimeValue endTime,
   TimeValue timeIncrement,
   short firstDataRefIndex,
   short lastDataRefIndex,
   long *retdataRefSkew
);
```

Parameters

trackH
: The track for the operation. Your application gets this identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

offsets
: A handle to an unlocked relocatable memory block allocated by your application. The function returns the time table for the track in this block.

startTime
: The first point of the track to be included in the time table. This time value is expressed in the movie's time coordinate system.

endTime
: The last point of the track to be included in the time table. This time value is expressed in the movie's time coordinate system.

timeIncrement
: The resolution of the time table. The values in a time table are for a points in the track, and these points are separated by the amount of time specified by this parameter. The time value is expressed in the movie's time coordinate system.

firstDataRefIndex
: An index to the first data reference for the track to be included in the time table. Set this parameter to -1 to include all data references for the `track`. Set this parameter to 1 to specify the first data reference for the track.

lastDataRefIndex
: An index to the last data reference for the track to be included in the time table. The value 1 specifies the first data reference for the track. If the value of the `firstDataRefIndex` parameter is -1, set this parameter to 0.

retdataRefSkew
: The offset to the next row of the time table, as a long integer. The next row contains values for the next data reference, as explained below. By adding the `value` of this parameter to an offset into the table, you get the offset to the corresponding point for the next data reference.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

Your application must allocate an unlocked relocatable memory block for the time table to be returned and pass a handle to it in the `offsets` parameter. The `MakeTrackTimeTable` (page 1382) function resizes the block to accommodate the time table it returns.

This time table is a two-dimensional array of long integers that is organized so that each row in the table contains values for one data reference. The first column in the table contains values for the time in the track specified by the `startTime` parameter, and each subsequent column contains values for the point in the track that is later by the value specified by the `timeIncrement` parameter. Each long integer value in the table specifies the offset, in bytes, from the beginning of the data reference for that point in the track. The number of columns in the table is equal to `(endTime - startTime) / timeIncrement`, rounded up. Because of alignment issues, this value is not always the same as the value of the `retdataRefSkew` parameter. If there are track edits for a track, they are reflected in the track's time table.

Special Considerations

When all the data for a movie has been transferred, your application must dispose of the time table created by this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

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` (page 259), `NewMovieFromProperties` (page 260), `NewMovieFromFile`, and `NewMovieFromHandle` (page 1400).

`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` (page 1384).

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
ExtractMovieAudioToAIFF
QTAudioExtractionPanel
QTExtractAndConvertToAIFF
SCAudioCompress
SimpleAudioExtraction

Declared In
`Movies.h`

MovieAudioExtractionEnd

Ends a movie audio extraction session.

```
OSStatus MovieAudioExtractionEnd (
   MovieAudioExtractionRef session
);
```

Parameters

session
The session object returned by `MovieAudioExtractionBegin` (page 1383).

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

Discussion
You must call this function when movie audio extraction is complete.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
ExtractMovieAudioToAIFF
QTAudioExtractionPanel
QTExtractAndConvertToAIFF
QTExtractAndConvertToMovieFile
SimpleAudioExtraction

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 1383).

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). See these constants:

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.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
ExtractMovieAudioToAIFF
QTAudioExtractionPanel
QTExtractAndConvertToAIFF
SCAudioCompress
SimpleAudioExtraction

Declared In
Movies.h

MovieAudioExtractionGetProperty

Gets a property of a movie audio extraction session.

```
OSStatus MovieAudioExtractionGetProperty (
   MovieAudioExtractionRef session,
   QTPropertyClass inPropClass,
   QTPropertyID inPropID,
   ByteCount inPropValueSize,
   QTPropertyValuePtr outPropValueAddress,
   ByteCount *outPropValueSizeUsed
);
```

Parameters

session

The session object returned by `MovieAudioExtractionBegin` (page 1383).

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'`. See these constants:

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.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

ExtractMovieAudioToAIFF

QTAudioExtractionPanel

SCAudioCompress

SimpleAudioExtraction

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 1383).

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'`. See these constants:

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.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
ExtractMovieAudioToAIFF

QTAudioExtractionPanel

SCAudioCompress

SimpleAudioExtraction

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 1383).

`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'`. See these constants:

`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.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
QTAudioExtractionPanel
QTExtractAndConvertToAIFF
QTExtractAndConvertToMovieFile
SCAudioCompress
SimpleAudioExtraction

Declared In
`Movies.h`

MovieExecuteWiredActions

Undocumented

```
OSErr MovieExecuteWiredActions (
   Movie theMovie,
   long flags,
   QTAtomContainer actions
);
```

Parameters

theMovie
A movie identifier. Your application obtains this identifier from such functions as NewMovie (page 259), NewMovieFromFile (page 1398), and NewMovieFromHandle (page 1400).

flags
Undocumented See these constants:
movieExecuteWiredActionDontExecute

actions
Undocumented

Return Value
You can access Movie Toolbox error returns through GetMoviesError (page 221) and GetMoviesStickyError (page 222), as well as in the function result. See Error Codes.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
Movies.h

MovieSearchText

Searches for text in a movie.

```
OSErr MovieSearchText (
   Movie theMovie,
   Ptr text,
   long size,
   long searchFlags,
   Track *searchTrack,
   TimeValue *searchTime,
   long *searchOffset
);
```

Parameters

theMovie
A movie identifier. Your application obtains this identifier from such functions as NewMovie (page 259), NewMovieFromFile (page 1398), and NewMovieFromHandle (page 1400).

text
The text to be searched for.

size
The size of the text.

searchFlags
: Flags (see below) that narrow the search process. See these constants:
  `searchTextDontGoToFoundTime`
  `searchTextDontHiliteFoundText`
  `searchTextOneTrackOnly`
  `searchTextEnabledTracksOnly`

searchTrack
: On return, a pointer to the found track.

searchTime
: On return, a pointer to the found time.

searchOffset
: On return, a pointer to the found offset to the text.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qttext
qttext.win

Declared In
`Movies.h`

NewActionsUPP

Allocates a Universal Procedure Pointer for ActionsProc.

```
ActionsUPP NewActionsUPP (
   ActionsProcPtr userRoutine
);
```

Parameters

userRoutine
: A pointer to your application-defined function.

Return Value
A new UPP; see `Universal Procedure Pointers`.

Discussion
This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes
Introduced in QuickTime 4.1. Replaces `NewActionsProc`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

NewDoMCActionUPP

Allocates a Universal Procedure Pointer for the DoMCActionProc callback.

```
DoMCActionUPP NewDoMCActionUPP (
   DoMCActionProcPtr userRoutine
);
```

Parameters

`userRoutine`

A pointer to your application-defined function.

Return Value
A new UPP; see `Universal Procedure Pointers`.

Discussion
This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes
Introduced in QuickTime 4.1. Replaces `NewDoMCActionProc`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

NewGetMovieUPP

Allocates a Universal Procedure Pointer for the GetMovieProc callback.

```
GetMovieUPP NewGetMovieUPP (
   GetMovieProcPtr userRoutine
);
```

Parameters

`userRoutine`

A pointer to your application-defined function.

Return Value
A new UPP; see `Universal Procedure Pointers`.

Discussion
This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes
Introduced in QuickTime 4.1. Replaces `NewGetMovieProc`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

NewMovieController

Locates a movie controller component and assigns a movie to that controller.

```
ComponentInstance NewMovieController (
   Movie theMovie,
   const Rect *movieRect,
   long someFlags
);
```

Parameters

theMovie

The movie to be associated with the movie controller.

movieRect

A pointer to the `Rect` structure that is to define the display boundaries of the movie and its controller.

someFlags

Contains flags (see below) that control the operation. If you set these flags to 0, the movie controller component centers the movie in the rectangle specified by the `movieRect` parameter and scales the movie to fit in that rectangle. The control portion of the controller is also placed within that rectangle. You may control how the movie and the control are drawn by setting one or more flags to 1. See these constants:

`mcTopLeftMovie`

`mcScaleMovieToFit`

`mcWithBadge`

`mcNotVisible`

`mcWithFrame`

Return Value
The ID of the new controller.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CarbonQTGraphicImport

MakeEffectMovie

qteffects.win

qtstreamsplicer

qtstreamsplicer.win

Declared In
`Movies.h`

NewMovieDrawingCompleteUPP

Allocates a Universal Procedure Pointer for the MovieDrawingCompleteProc callback.

```
MovieDrawingCompleteUPP NewMovieDrawingCompleteUPP (
   MovieDrawingCompleteProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your application-defined function.

Return Value

A new UPP; see `Universal Procedure Pointers`.

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes

Introduced in QuickTime 4.1. Replaces `NewMovieDrawingCompleteProc`.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

ASCIIMoviePlayerSample

ASCIIMoviePlayerSample for Windows

MovieGWorlds

OpenGLMovieQT

VideoProcessing

Declared In

`Movies.h`

NewMovieExecuteWiredActionsUPP

Allocates a Universal Procedure Pointer for the MovieExecuteWiredActionsProc callback.

```
MovieExecuteWiredActionsUPP NewMovieExecuteWiredActionsUPP (
   MovieExecuteWiredActionsProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your application-defined function.

Return Value

A new UPP; see `Universal Procedure Pointers`.

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes

Introduced in QuickTime 4.1. Replaces `NewMovieExecuteWiredActionsProc`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

NewMovieForDataRefFromHandle

Creates a movie from a public movie handle, converting internal references to external references.

```
OSErr NewMovieForDataRefFromHandle (
   Movie *theMovie,
   Handle h,
   short newMovieFlags,
   Boolean *dataRefWasChanged,
   Handle dataRef,
   OSType dataRefType
);
```

Parameters

theMovie
A pointer to a field that is to receive the new movie's identifier. If the function cannot load the movie, the returned identifier is set to `NIL`.

h
A handle to the movie resource from which the movie is to be loaded.

newMovieFlags
Constants (see below) that control characteristics of the new movie. Set unused flags to 0. See these constants:

`newMovieActive`

`newMovieDontResolveDataRefs`

`newMovieDontAskUnresolvedDataRefs`

dataRefWasChanged
A pointer to a Boolean value. The toolbox sets the value to TRUE if any references were changed. Pass `NIL` if you don't want to receive this information.

dataRef
A data reference to the storage from which the movie was retrieved.

dataRefType
The data reference type. See `Data References`.

Return Value
If the Movie Toolbox cannot completely resolve all data references, it sets the current error value to `couldNotResolveDataRef`. You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
This function creates a movie from a public movie handle in the same way as `NewMovieFromHandle` (page 1400), but with one difference. If the public handle contains internal media data references, the function can convert them to external references, as specified by `dataRef` and `dataRefType`. No other data references are changed.

Version Notes
Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`Movies.h`

NewMovieFromDataFork

Retrieves a movie that is stored anywhere in the data fork of a specified Macintosh file.

```
OSErr NewMovieFromDataFork (
   Movie *theMovie,
   short fRefNum,
   long fileOffset,
   short newMovieFlags,
   Boolean *dataRefWasChanged
);
```

Parameters

theMovie
A pointer to a field that is to receive the new movie's identifier. If the function cannot load the movie, the returned identifier is set to `NIL`.

fRefNum
A file reference number to a file that is already open.

fileOffset
The starting file offset of the atom in the data fork of the file specified by the `fRefNum` parameter.

newMovieFlags
Flags (see below) that control characteristics of the new movie. See these constants:
`newMovieActive`
`newMovieDontResolveDataRefs`
`newMovieDontAskUnresolvedDataRefs`

dataRefWasChanged
A pointer to a Boolean value. The Movie Toolbox sets the value to TRUE if any of the movie's data references were changed. Use `UpdateMovieResource` (page 1503) to preserve these changes. If you do not want to receive this information, set the `dataRefWasChanged` parameter to `NIL`.

Return Value
If the Movie Toolbox cannot completely resolve all data references, it sets the current error value to `couldNotResolveDataRef`. You can access error returns such as this through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Special Considerations

The Movie Toolbox automatically sets the movie's graphics world based on the current graphics port. Be sure that your application's graphics port is valid before you call this function, even if the movie is sound-only; you can use `GetGWorld` to check for a valid port, or you can use `NewGWorld` to create a port. The graphics port must remain valid for the life of the movie or until you set another valid graphics port for the movie using `SetMovieGWorld` (page 290).

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

NewMovieFromDataFork64

Provides a 64-bit version of NewMovieFromDataFork.

```
OSErr NewMovieFromDataFork64 (
   Movie *theMovie,
   long fRefNum,
   const wide *fileOffset,
   short newMovieFlags,
   Boolean *dataRefWasChanged
);
```

Parameters

theMovie
A pointer to a field that is to receive the new movie's identifier. If the function cannot load the movie, the returned identifier is set to `NIL`.

fRefNum
A file reference number to a file that is already open.

fileOffset
A pointer to the starting file offset of the atom in the data fork of the file specified by the `fRefNum` parameter.

newMovieFlags
Flags (see below) that control characteristics of the new movie. See these constants:
`newMovieActive`
`newMovieDontResolveDataRefs`
`newMovieDontAskUnresolvedDataRefs`

dataRefWasChanged
A pointer to a Boolean value. The Movie Toolbox sets the value to TRUE if any of the movie's data references were changed. Use `UpdateMovieResource` (page 1503) to preserve these changes. If you do not want to receive this information, set the `dataRefWasChanged` parameter to `NIL`.

Return Value
If the Movie Toolbox cannot completely resolve all data references, it sets the current error value to `couldNotResolveDataRef`. You can access error returns such as this through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Special Considerations

The Movie Toolbox automatically sets the movie's graphics world based on the current graphics port. Be sure that your application's graphics port is valid before you call this function, even if the movie is sound-only; you can use `GetGWorld` to check for a valid port, or you can use `NewGWorld` to create a port. The graphics port must remain valid for the life of the movie or until you set another valid graphics port for the movie using `SetMovieGWorld` (page 290).

Version Notes

Introduced in QuickTime 4. Superseded in QuickTime 6 by `NewMovieFromStorageOffset` (page 1402).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

NewMovieFromDataRef

Creates a movie from any device with a corresponding data handler.

```
OSErr NewMovieFromDataRef (
   Movie *m,
   short flags,
   short *id,
   Handle dataRef,
   OSType dataRefType
);
```

Parameters

m

A pointer to a field that is to receive the new movie's identifier. If the function cannot load the movie, the returned identifier is set to `NIL`.

flags

Flags (see below) that control the operation of this function. Be sure to set unused flags to 0. See these constants:

`newMovieActive`

`newMovieDontResolveDataRefs`

`newMovieDontAskUnresolvedDataRefs`

id

A pointer to the field that specifies the resource containing the movie data that is to be loaded. If the field referred to by the `id` parameter is set to 0, the Movie Toolbox loads the first movie resource it finds in the specified file. The toolbox then returns the movie's resource ID number in the field referred to by the `id` parameter. An enumerated constant (see below) is available. See these constants:

`movieInDataForkResID`

dataRef

The default data reference. This parameter contains a handle to the information that identifies the file to be used to resolve any data references and as a starting point for any Alias Manager searches. The type of information stored in the handle depends upon the value of the `dataRefType` parameter. For example, if your application is loading the movie from a file, you would refer to the file's alias in this parameter and set the `dataRefType` parameter to `rAliasType`. If you do not want to identify a default data reference, set the parameter to `NIL`.

dataRefType

The type of data reference. If the data reference is an alias, you must set the parameter to `rAliasType`, indicating that the reference is an alias.

Return Value

If the Movie Toolbox cannot completely resolve all data references, it sets the current error value to `couldNotResolveDataRef`. You can access error returns such as this through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

This function is intended for use by specialized applications that need to instantiate movies from devices not visible to the file system. Most applications should continue to use `NewMovieFromFile` (page 1398). You are not restricted to instantiating a movie from a file stored on a Macintosh HFS volume. With this function, you can instantiate a movie from any device.

Special Considerations

The Movie Toolbox automatically sets the movie's graphics world based on the current graphics port. Be sure that your application's graphics port is valid before you call this function, even if the movie is sound-only; you can use `GetGWorld` to check for a valid port, or you can use `NewGWorld` to create a port. The graphics port must remain valid for the life of the movie or until you set another valid graphics port for the movie using `SetMovieGWorld` (page 290).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qtdataref

qtdataref.win

SlideShowImporter.win

ThreadsImporter

ThreadsImportMovie

Declared In

`Movies.h`

NewMovieFromFile

Creates a new movie in memory from a movie file or from any type of file for which QuickTime provides an import component (AIFF, JPEG, MPEG-4, etc).

```
OSErr NewMovieFromFile (
   Movie *theMovie,
   short resRefNum,
   short *resId,
   StringPtr resName,
   short newMovieFlags,
   Boolean *dataRefWasChanged
);
```

Parameters

`theMovie`

A pointer to a field that is to receive the new movie's identifier. If the function cannot load the movie, the returned identifier is set to `NIL`.

resRefNum
: The movie file from which the movie is to be loaded. Your application obtains this value from the `OpenMovieFile` (page 1416) function.

resId
: A pointer to a field that specifies the resource containing the movie data that is to be loaded. If the field referred to by the `resId` parameter is set to 0, the Movie Toolbox loads the first movie resource it finds in the specified file. The Movie Toolbox then returns the movie's resource ID number in the field referred to by the `resId` parameter. An enumerated constant (see below) is available. See these constants:

    `movieInDataForkResID`

resName
: A pointer to a character string that is to receive the name of the movie resource that is loaded. If you set the `resName` parameter to `NIL`, the Movie Toolbox does not return the resource name.

newMovieFlags
: Flags (see below) that control the operation of `NewMovieFromFile`. Be sure to set unused flags to 0. See these constants:

    `newMovieActive`

    `newMovieDontResolveDataRefs`

    `newMovieDontAskUnresolvedDataRefs`

dataRefWasChanged
: A pointer to a Boolean value. The Movie Toolbox sets the value to TRUE if any references were changed. Use `UpdateMovieResource` (page 1503) to preserve these changes. Set this parameter to `NIL` if you don't want to receive this information. See `NewMovieTrack` (page 1628) for more information about data references.

Return Value
If the Movie Toolbox cannot completely resolve all data references, it sets the current error value to `couldNotResolveDataRef`. You can access error returns such as this through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
The Movie Toolbox sets many movie characteristics to default values. If you want to change these defaults, your application must call other Movie Toolbox functions. For example, the Movie Toolbox sets the movie's graphics world to the one that is active when you call `NewMovieFromFile`. To change the graphics world for the new movie, your application should use `SetMovieGWorld` (page 290).

The following is an example of using this function:

```
// NewMovieFromFile coding example
// See "Discovering QuickTime," page 385
Movie MyGetMovie (void)
{
    OSErr                   nErr;
    SFTypeList              types ={MovieFileType, 0, 0, 0};
    StandardFileReply       sfr;
    Movie                   movie =NIL;
    short                   nFileRefNum;
    StandardGetFilePreview(NIL, 1, types, &sfr);
    if (sfr.sfGood) {
        nErr =OpenMovieFile(&sfr.sfFile, &nFileRefNum, fsRdPerm);
        if (nErr ==noErr) {
            short           nResID =0;          //We want the first movie.
            Str255          strName;
            Boolean         bWasChanged;

            nErr =NewMovieFromFile(&movie, nFileRefNum, &nResID, strName,
                                   newMovieActive, &bWasChanged);
            CloseMovieFile(nFileRefNum);
        }
    }
    return movie;
}
```

Special Considerations

The Movie Toolbox automatically sets the movie's graphics world based on the current graphics port. Be sure that your application's graphics port is valid before you call this function, even if the movie is sound-only; you can use `GetGWorld` to check for a valid port, or you can use `NewGWorld` to create a port. The graphics port must remain valid for the life of the movie or until you set another valid graphics port for the movie using `SetMovieGWorld` (page 290).

Special Considerations

This function works with some files that don't contain movie resources. When it encounters a file that does not contain a movie resource, it tries to find a movie import component that can understand the data and create a movie. It also works for MPEG, uLaw (.AU), and Wave (.WAV) file types. In some cases, the data in a file is already sufficiently well formatted for QuickTime or its components to understand. For example, the AIFF movie data import component can understand AIFF sound files and import the sound data into a QuickTime movie.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

MakeEffectMovie

vrmakepano

vrmakepano.win

vrscript

vrscript.win

Declared In

`Movies.h`

NewMovieFromHandle

Creates a movie in memory from a movie resource or a handle you obtained from PutMovieIntoHandle.

```
OSErr NewMovieFromHandle (
   Movie *theMovie,
   Handle h,
   short newMovieFlags,
   Boolean *dataRefWasChanged
);
```

Parameters

theMovie
A pointer to a field that is to receive the new movie's identifier. If the function cannot load the movie, the returned identifier is set to `NIL`.

h
A handle to the movie resource from which the movie is to be loaded.

newMovieFlags
Flags (see below) that control the operation of `NewMovieFromHandle`. Be sure to set unused flags to 0. See these constants:

`newMovieActive`

`newMovieDontResolveDataRefs`

`newMovieDontAskUnresolvedDataRefs`

dataRefWasChanged
A pointer to a Boolean value. The toolbox sets the value to TRUE if any references were changed. Set the `dataRefWasChanged` parameter to `NIL` if you don't want to receive this information.

Return Value
If the Movie Toolbox cannot completely resolve all data references, it sets the current error value to `couldNotResolveDataRef`. You can access error returns such as this through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
The Movie Toolbox sets many movie characteristics to default values. If you want to change these defaults, your application must call other Movie Toolbox functions. For example, the Movie Toolbox sets the movie's graphics world to the one that is active when you call `NewMovieFromHandle`. To change the graphics world for the new movie, your application should use `SetMovieGWorld` (page 290).

Special Considerations

The Movie Toolbox automatically sets the movie's graphics world based on the current graphics port. Be sure that your application's graphics port is valid before you call this function, even if the movie is sound-only; you can use `GetGWorld` to check for a valid port, or you can use `NewGWorld` to create a port. The graphics port must remain valid for the life of the movie or until you set another valid graphics port for the movie using `SetMovieGWorld` (page 290).

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
ExtractMovieAudioToAIFF

QTAudioExtractionPanel

QTExtractAndConvertToAIFF

QTExtractAndConvertToMovieFile

ThreadsExportMovie

Declared In
`Movies.h`

NewMovieFromScrap

Creates a movie from the contents of the scrap.

```
Movie NewMovieFromScrap (
   long newMovieFlags
);
```

Parameters

`newMovieFlags`

Flags (see below) that control the operation of the `NewMovieFromScrap` function. Be sure to set unused flags to 0. See these constants:

`newMovieActive`

`newMovieDontResolveDataRefs`

`newMovieDontAskUnresolvedDataRefs`

Return Value
The identifier for the new movie. If `NewMovieFromScrap` fails, or if there is no movie in the scrap, the returned identifier is set to `NIL`. You can use `GetMoviesError` (page 221) to obtain the error result, or `noErr` if there was no error. See `Error Codes`.

Special Considerations

The Movie Toolbox automatically sets the movie's graphics world based on the current graphics port. Be sure that your application's graphics port is valid before you call this function, even if the movie is sound-only; you can use `GetGWorld` to check for a valid port, or you can use `NewGWorld` to create a port. The graphics port must remain valid for the life of the movie or until you set another valid graphics port for the movie using `SetMovieGWorld` (page 290).

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

NewMovieFromStorageOffset

Creates a new movie based on the offset to data in a storage container.

```
OSErr NewMovieFromStorageOffset (
   Movie *theMovie,
   DataHandler dh,
   const wide *fileOffset,
   short newMovieFlags,
   Boolean *dataRefWasChanged
);
```

Parameters

theMovie

A pointer to a field that is to receive the new movie's identifier. If the function cannot load the movie, the returned identifier is set to `NIL`

dh

The data handler component that was returned by `CreateMovieStorage` (page 1318). The data handler's file must be open.

fileOffset

A pointer to the location of the movie data in the storage location specified by the `dh` parameter. Unlike `NewMovieFromDataFork` and NewMovieFromDataFork64, there is no special meaning to a file offset of -1.

newMovieFlags

Constants (see below) that control characteristics of the new movie. See these constants:

`newMovieActive`

`newMovieDontResolveDataRefs`

`newMovieDontAskUnresolvedDataRefs`

dataRefWasChanged

A pointer to a Boolean value. The Movie Toolbox sets the value to TRUE if any of the movie's data references were changed. Use `UpdateMovieInStorage` (page 1503) to preserve these changes. If you do not want to receive this information, pass `NIL`.

Return Value

If the Movie Toolbox cannot completely resolve all data references, it sets the current error value to `couldNotResolveDataRef`. You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

This function serves the same purpose for data handlers as `NewMovieFromDataFork64` (page 1396) does for movie file references. The API reads the `'moov'` resource found at `fileOffset` and creates a Movie. The data handler parameter should be an open data handler component instance for the storage holding the `'moov'` resource. The `newMovieFlags` and `dataRefWasChanged` parameters are interpreted identically to those same parameters in NewMovieFromDataFork64.

If you are writing a custom data handler, make sure it implements `DataHGetDataRef` (page 780). Also implement `DataHScheduleData64` (page 807) and `DataHGetFileSize64` (page 785), or `DataHScheduleData` (page 806) and `DataHGetFileSize` (page 784) if the data handler does not support 64-bit file offsets.

Special Considerations

The Movie Toolbox automatically sets the movie's graphics world based on the current graphics port. Be sure that your application's graphics port is valid before you call this function, even if the movie is sound-only; you can use `GetGWorld` to check for a valid port, or you can use `NewGWorld` to create a port. The graphics port must remain valid for the life of the movie or until you set another valid graphics port for the movie using `SetMovieGWorld` (page 290).

Version Notes

Introduced in QuickTime 6. Supersedes `NewMovieFromDataFork64` (page 1396).

Availability

Available in Mac OS X v10.2 and later.

Declared In

`Movies.h`

NewMovieFromUserProc

Creates a movie from data that you provide.

```
OSErr NewMovieFromUserProc (
   Movie *m,
   short flags,
   Boolean *dataRefWasChanged,
   GetMovieUPP getProc,
   void *refCon,
   Handle defaultDataRef,
   OSType dataRefType
);
```

Parameters

m

A pointer to a field that is to receive the new movie's identifier. If the function cannot load the movie, the returned identifier is set to `NIL`.

flags

Flags (see below) that control the operation of the `NewMovieFromUserProc` function. Be sure to set unused flags to 0. See these constants:

`newMovieActive`

`newMovieDontResolveDataRefs`

`newMovieDontAskUnresolvedDataRefs`

dataRefWasChanged

A pointer to a Boolean value. The Toolbox sets the value to TRUE if any references were changed. Use `UpdateMovieResource` (page 1503) to preserve these changes. Set the `dataRefWasChanged` parameter to `NIL` if you don't want to receive this information.

getProc

A Universal Procedure Pointer that accesses a `GetMovieProc` callback, which is responsible for providing the movie data to the Movie Toolbox.

refCon

A reference constant (defined as a void pointer). This is the same value you provided to the Movie Toolbox when you called `NewMovieFromUserProc`. Use this parameter to point to a data structure containing any information your callback needs.

defaultDataRef

The default data reference. This parameter contains a handle to the information that identifies the file to be used to resolve any data references and as a starting point for any Alias Manager searches. The type of information stored in the handle depends upon the value of the `dataRefType` parameter. For example, if your application is loading the movie from a file, you would refer to the file's alias in the `defaultDataRef` parameter, and set the `dataRefType` parameter to `rAliasType`. If you don't want to identify a default data reference, set the parameter to `NIL`.

dataRefType

The type of data reference. If the data reference is an alias, you must set the parameter to `rAliasType`, indicating that the reference is an alias.

Return Value

If the Movie Toolbox cannot completely resolve all data references, it sets the current error value to `couldNotResolveDataRef`. You can access error returns such as this through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

Normally, when a movie is loaded from a file (for example, by means of `NewMovieFromFile` (page 1398)), the Movie Toolbox uses that file as the default data reference. Since this function does not require a file specification, your application should specify the file to be used as the default data reference using the `defaultDataRef` and `dataRefType` parameters.

Special Considerations

The Movie Toolbox automatically sets the movie's graphics world based on the current graphics port. Be sure that your application's graphics port is valid before you call this function, even if the movie is sound-only; you can use `GetGWorld` to check for a valid port, or you can use `NewGWorld` to create a port. The graphics port must remain valid for the life of the movie or until you set another valid graphics port for the movie using `SetMovieGWorld` (page 290).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

NewMoviePrePrerollCompleteUPP

Allocates a Universal Procedure Pointer for the MoviePrePrerollCompleteProc callback.

```
MoviePrePrerollCompleteUPP NewMoviePrePrerollCompleteUPP (
   MoviePrePrerollCompleteProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your application-defined function.

Return Value

A new UPP; see `Universal Procedure Pointers`.

Discussion
This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes
Introduced in QuickTime 4.1. Replaces `NewMoviePrePrerollCompleteProc`.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
vrscript

vrscript.win

Declared In
`Movies.h`

NewMoviePreviewCallOutUPP

Allocates a Universal Procedure Pointer for the MoviePreviewCallOutProc callback.

```
MoviePreviewCallOutUPP NewMoviePreviewCallOutUPP (
   MoviePreviewCallOutProcPtr userRoutine
);
```

Parameters
`userRoutine`

A pointer to your application-defined function.

Return Value
A new UPP; see `Universal Procedure Pointers`.

Discussion
This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes
Introduced in QuickTime 4.1. Replaces `NewMoviePreviewCallOutProc`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

NewMovieProgressUPP

Allocates a Universal Procedure Pointer for the MovieProgressProc callback.

```
MovieProgressUPP NewMovieProgressUPP (
   MovieProgressProcPtr userRoutine
);
```

Parameters
`userRoutine`

A pointer to your application-defined function.

Return Value
A new UPP; see `Universal Procedure Pointers`.

Discussion
This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes
Introduced in QuickTime 4.1. Replaces `NewMovieProgressProc`.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CIVideoDemoGL

qtdataexchange

qtdataexchange.win

ThreadsExportMovie

ThreadsImportMovie

Declared In
`Movies.h`

NewMovieRgnCoverUPP

Allocates a Universal Procedure Pointer for the MovieRgnCoverProc callback.

```
MovieRgnCoverUPP NewMovieRgnCoverUPP (
   MovieRgnCoverProcPtr userRoutine
);
```

Parameters

`userRoutine`
: A pointer to your application-defined function.

Return Value
A new UPP; see `Universal Procedure Pointers`.

Discussion
This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes
Introduced in QuickTime 4.1. Replaces `NewMovieRgnCoverProc`.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
vrmovies

vrmovies.win

vrscript

vrscript.win

Declared In
`Movies.h`

NewMoviesErrorUPP

Allocates a Universal Procedure Pointer for the MoviesErrorProc callback.

```
MoviesErrorUPP NewMoviesErrorUPP (
   MoviesErrorProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your application-defined function.

Return Value
A new UPP; see `Universal Procedure Pointers`.

Version Notes
Introduced in QuickTime 4.1. Replaces `NewMoviesErrorProc`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

NewQTCallBackUPP

Allocates a Universal Procedure Pointer for the QTCallBackProc callback.

```
QTCallBackUPP NewQTCallBackUPP (
   QTCallBackProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your application-defined function.

Return Value
A new UPP; see `Universal Procedure Pointers`.

Discussion
This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes
Introduced in QuickTime 4.1. Replaces `NewQTCallBackProc`.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qtbigscreen

qtbigscreen.win

SimpleCocoaMovie

SimpleCocoaMovieQT

Declared In
`Movies.h`

NewQTEffectListFilterUPP

Allocates a Universal Procedure Pointer for the QTEffectListFilterProc callback.

```
QTEffectListFilterUPP NewQTEffectListFilterUPP (
   QTEffectListFilterProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to a QTEffectListFilterProc callback.

Return Value
A new UPP; see Universal Procedure Pointers.

Version Notes
Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.2 and later.

Declared In
Movies.h

NewQTNextTaskNeededSoonerCallbackUPP

Allocates a Universal Procedure Pointer for the QTNextTaskNeededSoonerCallbackProc callback.

```
QTNextTaskNeededSoonerCallbackUPP NewQTNextTaskNeededSoonerCallbackUPP (
   QTNextTaskNeededSoonerCallbackProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to a QTNextTaskNeededSoonerCallbackProc callback.

Return Value
A new UPP; see Universal Procedure Pointers.

Version Notes
Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
qtshellCEvents

qtshellCEvents.win

VideoProcessing

Declared In
Movies.h

NewQTSyncTaskUPP

Allocates a Universal Procedure Pointer for the QTSyncTaskProc callback.

```
QTSyncTaskUPP NewQTSyncTaskUPP (
   QTSyncTaskProcPtr userRoutine
);
```

Parameters

userRoutine
A pointer to your application-defined function.

Return Value
A new UPP; see `Universal Procedure Pointers`.

Discussion
This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes
Introduced in QuickTime 4.1. Replaces `NewQTSyncTaskProc`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

NewSprite

Creates a new sprite in a specified sprite world.

```
OSErr NewSprite (
   Sprite *newSprite,
   SpriteWorld itsSpriteWorld,
   ImageDescriptionHandle idh,
   Ptr imageDataPtr,
   MatrixRecord *matrix,
   Boolean visible,
   short layer
);
```

Parameters

newSprite
A pointer to field that is to receive the new sprite's identifier. On return, this field contains the identifier of the newly created sprite.

itsSpriteWorld
The sprite world with which the new sprite should be associated.

idh
A handle to an `ImageDescription` structure of the sprite's image.

imageDataPtr
A pointer to the sprite's image data.

matrix
A pointer to the sprite's `MatrixRecord` structure. If you pass `NIL`, an identity matrix is assigned to the sprite.

visible

Specifies whether the sprite is visible.

layer

The sprite's layer. Sprites with lower layer values appear in front of sprites with higher layer values. If you want to create a sprite that is drawn to the background graphics world, you should specify the constant `kBackgroundSpriteLayerNum` for the `layer` parameter.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

The visible parameter, the `layer` parameter, and the `newSprite` and `itsSpriteWorld` parameters are required. You can defer assigning image data to the sprite by passing `NIL` for both the `idh` and `imageDataPtr` parameters. If you choose to defer assigning image data, you must call `SetSpriteProperty` (page 1491) to assign the image description handle and image data to the sprite before the next call to `SpriteWorldIdle` (page 1502).

Special Considerations

The caller owns the image description handle and the image data pointer; it is the caller's responsibility to dispose of them after it disposes of a sprite.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Desktop Sprites

DesktopSprites

DesktopSprites.win

Declared In

`Movies.h`

NewSpriteWorld

Creates a new sprite world.

```
OSErr NewSpriteWorld (
   SpriteWorld *newSpriteWorld,
   GWorldPtr destination,
   GWorldPtr spriteLayer,
   RGBColor *backgroundColor,
   GWorldPtr background
);
```

Parameters

newSpriteWorld

A pointer to a field that is to receive the new sprite world's identifier. On return, this field contains the identifier for the newly created sprite world.

destination
: A pointer to a `CGrafPort` structure that defines the graphics world to be used as the destination.

spriteLayer
: A pointer to a `CGrafPort` structure that defines the graphics world to be used as the sprite layer.

backgroundColor
: A pointer to an `RGBColor` structure that defines the color to be used as the background color. If you pass a background graphics world to this function by setting the `background` parameter, you can set this parameter to `NIL`.

background
: A pointer to a `CGrafPort` structure that defines the graphics world to be used as the background. If you pass a background color to this function by setting the `backgroundColor` parameter, you can set this parameter to `NIL`.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
You call this function to create a new sprite world with associated destination and sprite layer graphics worlds, and either a background color or a background graphics world. Once created, you can manipulate the sprite world and add sprites to it using other sprite Movie Toolbox functions.

The `newSpriteWorld`, destination, and `spriteLayer` parameters are all required. You should specify a background color, a background graphics world, or both. You should not pass `NIL` for both parameters. If you specify both a background graphics world and a background color, the sprite world is filled with the background color before the background sprites are drawn. If no background color is specified, black is the default. If you specify a background graphics world, it should have the same dimensions and depth as the graphics world specified by `spriteLayer`. If you draw to the graphics worlds associated with a sprite world using standard QuickDraw and QuickTime functions, your drawing is erased by the sprite world's background color. The sprite world created by this function has an identity matrix and does not have a clip shape.

Here is an example of creating a sprite world:

```
// NewSpriteWorld coding example
// See "Discovering QuickTime," page 166
GWorldPtr       pSpritePlane =NIL;
SpriteWorld     spriteWorld =NIL;
Rect            rectBounce;
RGBColor        rgbcBackground;
void CreateSpriteStuff (Rect *pWndRect, CGrafPtr pMacWnd)
{
    OSErr       nErr;
    Rect        rect;
    // calculate the size of the destination
    rect =*pWndRect;
    OffsetRect(&rect, -rect.left, -rect.top);
    rectBounce =rect;
    InsetRect(&rectBounce, 16, 16);
    // create a sprite graphics world with a bit depth of 16
    NewGWorld(&pSpritePlane, 16, &rect, NIL, NIL, useTempMem);
    if (pSpritePlane ==NIL)
        NewGWorld(&pSpritePlane, 16, &rect, NIL, NIL, 0);
    if (pSpritePlane !=NIL) {
        LockPixels(pSpritePlane->
portPixMap);
```

```
        rgbcBackground.red =
        rgbcBackground.green =
        rgbcBackground.blue =0;
        // create a sprite world
        nErr =NewSpriteWorld(&spriteWorld, (CGrafPtr)pMacWnd,
            pSpritePlane, &rgbcBackground, NIL);
    }
}
```

Special Considerations

Before calling this function, you should lock the pixel maps of the sprite layer and background graphics worlds. These graphics worlds must remain valid and locked for the lifetime of the sprite world. The sprite world does not own the graphics worlds that are associated with it; it is the caller's responsibility to dispose of the graphics worlds when they are no longer needed.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Desktop Sprites

DesktopSprites

DesktopSprites.win

Declared In

`Movies.h`

NewTextMediaUPP

Allocates a Universal Procedure Pointer for the TextMediaProc callback.

```
TextMediaUPP NewTextMediaUPP (
   TextMediaProcPtr userRoutine
);
```

Parameters

`userRoutine`

A pointer to your application-defined function.

Return Value

A new UPP; see `Universal Procedure Pointers`.

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes

Introduced in QuickTime 4.1. Replaces `NewTextMediaProc`.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qttext

qttext.win

Declared In
`Movies.h`

NewTrackTransferUPP

Allocates a Universal Procedure Pointer for the TrackTransferProc callback.

```
TrackTransferUPP NewTrackTransferUPP (
   TrackTransferProcPtr userRoutine
);
```

Parameters

`userRoutine`
A pointer to your application-defined function.

Return Value
A new UPP; see `Universal Procedure Pointers`.

Discussion
This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes
Introduced in QuickTime 4.1. Replaces `NewTrackTransferProc`.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
MovieGWorlds

Declared In
`Movies.h`

NewTweenerDataUPP

Allocates a Universal Procedure Pointer for the TweenerDataProc callback.

```
TweenerDataUPP NewTweenerDataUPP (
   TweenerDataProcPtr userRoutine
);
```

Parameters

`userRoutine`
A pointer to your application-defined function.

Return Value
A new UPP; see `Universal Procedure Pointers`.

Discussion
This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes
Introduced in QuickTime 4.1. Replaces `NewTweenerDataProc`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

NewUserData

Creates a new user data structure.

```
OSErr NewUserData (
   UserData *theUserData
);
```

Parameters

theUserData

A pointer to a pointer to a new `UserDataRecord` structure.

Return Value
See `Error Codes`. Returns `noErr` if there is no error. If the function fails, `theUserData` is set to `NIL`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
AlwaysPreview
Graphic Import-Export
QTKitTimeCode
qttimecode
qttimecode.win

Declared In
`Movies.h`

NewUserDataFromHandle

Creates a new user data structure from a handle.

```
OSErr NewUserDataFromHandle (
   Handle h,
   UserData *theUserData
);
```

Parameters

h

A handle to the data structure specified in `theUserData`.

theUserData

A pointer to a pointer to a new `UserDataRecord` structure.

Return Value
See `Error Codes`. Returns `noErr` if there is no error. If the function fails, `theUserData` is set to `NIL`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
MungSaver

WhackedTV

Declared In
`Movies.h`

OpenMovieFile

Opens a specified movie file.

```
OSErr OpenMovieFile (
   const FSSpec *fileSpec,
   short *resRefNum,
   SInt8 permission
);
```

Parameters

`fileSpec`
A pointer to the `FSSpec` structure for the movie file to be opened.

`resRefNum`
A pointer to a field that is to receive the file reference number for the opened movie file. Your application must use this value when calling other Movie Toolbox functions that work with movie files. This reference number refers to the file fork that contains the movie resource. If the movie is stored in the data fork of the file, the returned reference number corresponds to the data fork.

`permission`
The permission level for the file (see below). If your application is only going to play the movie that is stored in the file, you can open the file with read permission. If you plan to add data to the file or change data in the file, you should open the file with write permission. See these constants:

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
Your application must open a movie file before reading movie data from it or writing movie data to it. You can open a movie file more than once; be sure to call `CloseMovieFile` (page 1311) once for each time you call this function. Note that opening the movie file with write permission does not prevent other applications from reading data from the movie file.

If the specified file has a resource fork, this function opens the resource fork and returns a file reference number to the resource fork. If the movie file does not have a resource fork (that is, it is a single-fork movie file), this function opens the data fork instead. In this case, your application cannot use `AddMovieResource` (page 1299) with the movie file.

The following is an example of using `OpenMovieFile`:

```
// OpenMovieFile coding example
// See "Discovering QuickTime," page 385
Movie MyGetMovie (void)
{
    OSErr                   nErr;
    SFTypeList              types ={MovieFileType, 0, 0, 0};
    StandardFileReply       sfr;
    Movie                   movie =NIL;
    short                   nFileRefNum;
    StandardGetFilePreview(NIL, 1, types, &sfr);
    if (sfr.sfGood) {
        nErr =OpenMovieFile(&sfr.sfFile, &nFileRefNum, fsRdPerm);
        if (nErr ==noErr) {
            short           nResID =0;          //We want the first movie.
            Str255          strName;
            Boolean         bWasChanged;

            nErr =NewMovieFromFile(&movie, nFileRefNum, &nResID, strName,
                                    newMovieActive, &bWasChanged);
            CloseMovieFile(nFileRefNum);
        }
    }
    return movie;
}
```

Version Notes
Introduced in QuickTime 3 or earlier. Superseded in QuickTime 6 by `OpenMovieStorage` (page 1417).

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
MakeEffectMovie
vrmakepano
vrmakepano.win
vrscript
vrscript.win

Declared In
`Movies.h`

OpenMovieStorage

Opens a data handler for movie storage.

```
OSErr OpenMovieStorage (
   Handle dataRef,
   OSType dataRefType,
   long flags,
   DataHandler *outDataHandler
);
```

Parameters

dataRef

A handle to a QuickTime data reference.

dataRefType

The data reference type. See `Data References`.

flags

A constant (see below) that determines the reading and writing capabilities of the data handler. See these constants:

`kDataHCanRead`

`kDataHCanWrite`

outDataHandler

A pointer to a field that is to receive the data handler for the opened movie file. Your application uses this value when calling other Movie Toolbox functions that work with movie files. If you pass `NIL`, the Movie Toolbox creates the movie storage but does not open it.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

This function is rarely used. It is an alternative to `OpenMovieFile` (page 1416).

Version Notes

Introduced in QuickTime 6.

Availability

Available in Mac OS X v10.2 and later.

Related Sample Code

CreateMovieFromReferences

QTCarbonShell

Declared In

`Movies.h`

PutMovieOnScrap

Places a movie into the Macintosh scrap.

```
OSErr PutMovieOnScrap (
   Movie theMovie,
   long movieScrapFlags
);
```

Parameters

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

movieScrapFlags

Flags (see below) that control the operation. Be sure to set unused flags to 0. See these constants:

`movieScrapDontZeroScrap`

`movieScrapOnlyPutMovie`

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

mdiplayer.win

mfc.win

Play Movie with Controller

simpleeditsdi.win

simpleplayersdi.win

Declared In

`Movies.h`

PutUserDataIntoHandle

Returns a handle to a user data structure.

```
OSErr PutUserDataIntoHandle (
   UserData theUserData,
   Handle h
);
```

Parameters

theUserData

The user data structure.

h

A handle to the `UserDataRecord` structure pointed to by the `theUserData` parameter.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
MungSaver
WhackedTV

Declared In
`Movies.h`

QTAddMovieError

Adds orthogonal errors to a movie's list of errors.

```
OSErr QTAddMovieError (
   Movie movieH,
   Component c,
   long errorCode,
   QTErrorReplacementPtr stringReplacements
);
```

Parameters

movieH

The movie to add the error to. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

c

An instance of the component that is adding the error. Your application obtains component instances by calling `OpenComponent` or `OpenDefaultComponent`.

errorCode

The error to be added.

stringReplacements

A pointer to a `QTErrorReplacementRecord` data structure that contains the list of strings to subsitute (in order) for "^1", "^2", etc.

Return Value
You can access the error return from this function through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
This routine is used to add orthogonal errors to a list of errors that will later be reported (at the end of an import or playback, for example). Errors are stored in `'qter'` resources within the component.

Version Notes
Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`Movies.h`

QTCopyAtom

Copies an atom and its children to a new atom container.

```
OSErr QTCopyAtom (
   QTAtomContainer container,
   QTAtom atom,
   QTAtomContainer *targetContainer
);
```

Parameters

container
The atom container that contains the atom to be copied.

atom
The atom to be copied. To duplicate the entire container, pass a value of `kParentAtomIsContainer` for the `atom` parameter.

targetContainer
A pointer to an uninitialized atom container data structure. On return, this parameter points to an atom container that contains a copy of the atom.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

The caller is responsible for disposing of the new atom container by calling `QTDisposeAtomContainer` (page 1427).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

addflashactions.win

qtwiredsprites

qtwiredsprites.win

SoftVideoOutputComponent

WiredSprites

Declared In

`Movies.h`

QTCopyAtomDataToHandle

Copies the specified leaf atom's data to a handle.

```
OSErr QTCopyAtomDataToHandle (
   QTAtomContainer container,
   QTAtom atom,
   Handle targetHandle
);
```

Parameters

container
: The atom container that contains the leaf atom.

atom
: The leaf atom whose data should be copied.

targetHandle
: A handle. On return, the handle contains the atom's data. The handle must not be locked. This function resizes the handle, if necessary.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

You call this function, passing an initialized handle, to retrieve a copy of a leaf atom's data.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qteffects.win

qtsprites.win

qtwiredactions

qtwiredsprites

qtwiredspritesjr

Declared In

`Movies.h`

QTCopyAtomDataToPtr

Copies the specified leaf atom's data to a buffer.

```
OSErr QTCopyAtomDataToPtr (
   QTAtomContainer container,
   QTAtom atom,
   Boolean sizeOrLessOK,
   long size,
   void *targetPtr,
   long *actualSize
);
```

Parameters

container
: The atom container that contains the leaf atom.

atom
: The leaf atom whose data should be copied.

sizeOrLessOK
: Specifies whether the function may copy fewer bytes than the number of bytes specified by the `size` parameter. The buffer may be larger than the amount of atom data if you set the `value` of this parameter to TRUE. You can determine the size of an atom's data by calling `QTGetAtomDataPtr` (page 1433).

size
: The length, in bytes, of the buffer pointed to by the `targetPtr` parameter.

targetPtr
: A pointer to a buffer. On return, the buffer contains the atom data.

actualSize
: A pointer to a long integer which, on return, contains the number of bytes copied to the buffer.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

You call this function, passing a data buffer, to retrieve a copy of a leaf atom's data. The buffer must be large enough to contain the atom's data.

Special Considerations

This function may move memory.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

vrbackbuffer.win

vrcursors

vrmakeobject

vrmovies

vrscript.win

Declared In

`Movies.h`

QTCountChildrenOfType

Returns the number of atoms of a given type in the child list of the specified parent atom.

```
short QTCountChildrenOfType (
   QTAtomContainer container,
   QTAtom parentAtom,
   QTAtomType childType
);
```

Parameters

container

The atom container that contains the parent atom.

parentAtom

The parent atom for this operation.

childType

The atom type for this operation. To retrieve the total number of atoms in the child list, set this parameter to 0.

Return Value

The number of atoms of a given type in the child list of the specified parent atom.

Discussion

You can call this function to determine the number of atoms of a specified type in a parent atom's child list. If the total number of atoms in the parent atom's child list is 0, the parent atom is a leaf atom.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Fiendishthngs

vrbackbuffer.win

vrcursors

vrmakeobject

vrmovies

Declared In

`Movies.h`

QTCreateStandardParameterDialog

Creates a dialog box that allows the user to choose an effect from the list of effects passed to the function.

```
OSErr QTCreateStandardParameterDialog (
   QTAtomContainer effectList,
   QTAtomContainer parameters,
   QTParameterDialogOptions dialogOptions,
   QTParameterDialog *createdDialog
);
```

Parameters

effectList

A list of the effects that the user can choose from. In most cases you should call `QTGetEffectsList` (page 1441) to generate this list. If you pass `NIL` in this parameter, the function calls `QTGetEffectsList` to retrieve the list of all currently installed effects; this list is then presented to the user.

parameters

An effect description containing the default parameter values for the effect. If the effect named in the parameter description is in effectlist, that effect is displayed when the dialog is first shown and its parameter values are set from the parameter description. Pass in an empty atom container to have the dialog box display the first effect in the list, set to its default parameters. On return, this atom container holds an effect description for the effect selected by the user, including the parameter settings. This effect description can then be added to the media of an effect track. You will need to add source atoms to this container for effects that require sources.

dialogOptions

Options (see below) that control the behavior of the dialog. See these constants:

`pdOptionsCollectOneValue`

`pdOptionsAllowOptionalInterpolations`

createdDialog

Returns a reference to the dialog box that is created by this function. You should pass this value only to `QTIsStandardParameterDialogEvent` (page 1448) and `QTDismissStandardParameterDialog` (page 1426).

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

This function creates and displays a standard parameter dialog box that allows the user to choose an effect from the list in the effectList parameter. The dialog box also allows the user to choose values for the parameters of the effect, to preview the effects as they choose and customize them, and to get more information about each effect. Your application must call the Mac OS function `WaitNextEvent` and `QTIsStandardParameterDialogEvent` (page 1448) to allow the user to interact with the dialog box that is shown. Note that the dialog box will remain hidden until the first event is processed by `QTIsStandardParameterDialogEvent`. At this point, the dialog box will be displayed. You can modify the default behavior of the dialog box that is created by calling `QTStandardParameterDialogDoAction` (page 1467).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

makeeffectslideshow

qteffects
qteffects.win
samplemakeeffectmovie
samplemakeeffectmovie.win

Declared In
`Movies.h`

QTCreateUUID

Creates a 128-bit universal unique ID number.

```
OSErr QTCreateUUID (
   QTUUID *outUUID,
   long creationFlags
);
```

Parameters

outUUID
A pointer to the new ID number.

creationFlags
Undocumented

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`Movies.h`

QTDismissStandardParameterDialog

Closes a standard parameter dialog box that was created using QTCreateStandardParameterDialog.

```
OSErr QTDismissStandardParameterDialog (
   QTParameterDialog createdDialog
);
```

Parameters

createdDialog
The reference to the standard parameters dialog box that is returned by `QTCreateStandardParameterDialog` (page 1424).

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

This function disposes of all memory associated with the dialog box.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

MakeEffectMovie

makeeffectslideshow

qteffects

qteffects.win

samplemakeeffectmovie.win

Declared In

`Movies.h`

QTDisposeAtomContainer

Disposes of an atom container.

```
OSErr QTDisposeAtomContainer (
   QTAtomContainer atomData
);
```

Parameters

`atomData`

The atom container to be disposed of.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

You can call this function to dispose of an atom container data structure that was created by `QTNewAtomContainer` (page 1451) or `QTCopyAtom` (page 1421).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qteffects

qteffects.win

qtwiredsprites

vrmakepano

WiredSprites

Declared In

`Movies.h`

QTDisposeTween

Disposes of a tween component instance.

```
OSErr QTDisposeTween (
   QTTweener tween
);
```

Parameters

tween

The tween to be disposed of.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Dimmer2Effect

Dimmer2Effect.win

Declared In

`Movies.h`

QTDoTween

Runs a tween component.

```
OSErr QTDoTween (
   QTTweener tween,
   TimeValue atTime,
   Handle result,
   long *resultSize,
   TweenerDataUPP tweenDataProc,
   void *tweenDataRefCon
);
```

Parameters

tween

The tween to be run.

atTime

A value that defines the time to run the tween.

result

A handle to the result of the tweening operation.

resultSize

A pointer to the size of the result.

tweenDataProc

A Universal Procedure Pointer that accesses a `TweenerDataProc` callback.

tweenDataRefCon

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

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Dimmer2Effect

Dimmer2Effect.win

Declared In

`Movies.h`

QTDoTweenPtr

Runs a tween component and returns values in a pointer rather than a handle.

```
OSErr QTDoTweenPtr (
   QTTweener tween,
   TimeValue atTime,
   Ptr result,
   long resultSize
);
```

Parameters

tween

A pointer to a `QTTweenerRecord` structure that designates the tween component to be run.

atTime

The time to run the tween.

result

A pointer to the result of the tween operation. The QuickTime atom container used to receive the tween result must be locked and its size must be large enough to contain the result.

resultSize

The size of the returned result.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`. Tween types that must allocate memory do not support this call; they return `codecUnimpErr`.

Discussion

This routine is an interrupt-safe version of `QTDoTween` (page 1428), which also runs a tween component. This call is not supported for sequence tweens; you should use interpolation tweens instead.

Version Notes

Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`Movies.h`

QTEqualUUIDs

Compares two 128-bit ID numbers.

```
Boolean QTEqualUUIDs (
   const QTUUID *uuid1,
   const QTUUID *uuid2
);
```

Parameters

uuid1
A pointer to one 128-bit number.

uuid2
A pointer to the other 128-bit number.

Return Value
Returns TRUE if the two numbers are equal, FALSE otherwise.

Version Notes
Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`Movies.h`

QTFindChildByID

Retrieves an atom by ID from the child list of the specified parent atom.

```
QTAtom QTFindChildByID (
   QTAtomContainer container,
   QTAtom parentAtom,
   QTAtomType atomType,
   QTAtomID id,
   short *index
);
```

Parameters

container
The atom container that contains the parent atom.

parentAtom
The parent atom for this operation.

atomType
The type of the atom to be retrieved.

id

The ID of the atom to be retrieved.

index

A pointer to an uninitialized short integer. On return, if the atom specified by the `id` parameter was found, the integer contains the atom's index. If you don't want this function to return the atom's index, set the value of the `index` parameter to `NIL`.

Return Value
The found atom.

Discussion
You call this function to search for and retrieve an atom by its type and ID from a parent atom's child list. The following code shows how you can use this function to insert a copy of container B's atoms as children of the `'abcd'` atom in container A:

```
// QTFindChildByID coding example
QTAtom targetAtom;
targetAtom =QTFindChildByID (containerA, kParentAtomIsContainer, 'abcd',
    1000, NIL);
FailOSErr (QTInsertChildren (containerA, targetAtom, containerB));
```

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
vrbackbuffer.win

vrcursors

vrmakeobject

vrmovies

vrscript.win

Declared In
`Movies.h`

QTFindChildByIndex

Retrieves an atom by index from the child list of the specified parent atom.

```
QTAtom QTFindChildByIndex (
   QTAtomContainer container,
   QTAtom parentAtom,
   QTAtomType atomType,
   short index,
   QTAtomID *id
);
```

Parameters

container

The atom container that contains the parent atom.

parentAtom
: The parent atom for this operation.

atomType
: The type of the atom to be retrieved.

index
: The index of the atom to be retrieved.

id
: A pointer to an uninitialized `QTAtomID` data structure. On return, if the atom specified by index was found, the `QTAtomID` data structure contains the atom's ID. If you don't want this function to return the atom's ID, set the `value` of the `id` parameter to `NIL`.

Return Value
The found atom.

Discussion
You call this function to search for and retrieve an atom by its type and index within that type from a parent atom's child list. The following code illustrates one way to use it:

```
// QTFindChildByIndex coding example
if ((propertyAtom =QTFindChildByIndex (sprite, kParentAtomIsContainer,
    kSpritePropertyImageIndex, 1, NIL)) ==0)
    FailOSErr (QTInsertChild (sprite, kParentAtomIsContainer,
        kSpritePropertyImageIndex, 1, 1, sizeof(short),&imageIndex,
        NIL));
```

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qteffects
qteffects.win
qtwiredactions
qtwiredsprites
qtwiredspritesjr

Declared In
`Movies.h`

QTGetAccessKeys

Returns all the application and system access keys of a specified access key type.

```
OSErr QTGetAccessKeys (
   Str255 accessKeyType,
   long flags,
   QTAtomContainer *keys
);
```

Parameters

accessKeyType

The type of access keys to return.

flags

Unused; must be set to 0.

keys

A pointer to a QT atom container that contains atoms of type `kAccessKeyAtomType` at the top level. These atoms contain the keys. If there are no access keys of the specified type, the function returns an empty QT atom container.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

In the QT atom container, application keys, which are more likely to be the ones an application needs, appear before system keys. You can get the key values by using QT atom functions.

Special Considerations

When your application is done with the QT atom container, it must dispose of it by calling `QTDisposeAtomContainer` (page 1427).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

QTGetAtomDataPtr

Retrieves a pointer to the atom data for a specified leaf atom.

```
OSErr QTGetAtomDataPtr (
   QTAtomContainer container,
   QTAtom atom,
   long *dataSize,
   Ptr *atomData
);
```

Parameters

container

The atom container that contains the leaf atom.

atom

The leaf atom whose data should be retrieved.

dataSize
: On return, contains a pointer to the length, in bytes, of the leaf atom's data.

atomData
: On return, contains a pointer to the leaf atom's data.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

You call this function in retrieve a pointer to a leaf atom's data so that you can access the data directly.

Special Considerations

To ensure that the pointer returned in the `atomData` parameter will remain valid if memory is moved, you should call `QTLockContainer` (page 1449) before you call this function. If you call `QTLockContainer`, you should call `QTUnlockContainer` (page 1469) when you have finished using the `atomData` pointer. If you pass a locked atom container to a function that resizes atom containers, the function returns an error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

MakeEffectMovie

qteffects.win

SimpleVideoOut

vrscript

vrscript.win

Declared In

`Movies.h`

QTGetAtomParent

Gets the parent of a QT atom.

```
QTAtom QTGetAtomParent (
   QTAtomContainer container,
   QTAtom childAtom
);
```

Parameters

container
: A QT atom container.

childAtom
: A QT child atom in the container.

Return Value

On return, the parent of the child atom.

Version Notes

Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

QTGetAtomTypeAndID

Retrieves an atom's type and ID.

```
OSErr QTGetAtomTypeAndID (
   QTAtomContainer container,
   QTAtom atom,
   QTAtomType *atomType,
   QTAtomID *id
);
```

Parameters

container
The atom container that contains the atom.

atom
The atom whose type and ID should be retrieved.

atomType
A pointer to an atom type. On return, this parameter points to the type of the specified atom. You can pass `NIL` for this parameter if you don't need this information.

id
A pointer to an atom ID. On return, this parameter points to the ID of the specified atom. You can pass `NIL` for this parameter if you don't need this information.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
vrmakeobject
vrmakepano
VRMakePano Library
vrmakepano.win
vrscript.win

Declared In
`Movies.h`

QTGetDataHandlerDirectoryDataReference

Returns a new data reference to the parent directory of the storage location associated with a data handler instance.

```
OSErr QTGetDataHandlerDirectoryDataReference (
   DataHandler dh,
   UInt32 flags,
   Handle *outDataRef,
   OSType *outDataRefType
);
```

Parameters

dh

A data handler component instance that is associated with a file.

flags

Currently not used; pass 0.

outDataRef

A pointer to a handle in which the newly created alias data reference is returned.

outDataRefType

A pointer to memory in which the `OSType` of the newly created data reference is returned.

Return Value

See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error. Returns `paramErr` if either of the output parameters was `NIL`.

Discussion

This function creates a new data reference that points at the parent directory of the storage location associated to the data handler instance.

Version Notes

Introduced in QuickTime 6.4.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`Movies.h`

QTGetDataHandlerFullPathCFString

Returns the full pathname of the storage location associated with a data handler.

```
OSErr QTGetDataHandlerFullPathCFString (
   DataHandler dh,
   QTPathStyle style,
   CFStringRef *outPath
);
```

Parameters

dh

A data handler component instance that is associated with a file.

style

A constant (see below) that identifies the syntax of the pathname. See these constants:

`kQTNativeDefaultPathStyle`

`kQTPOSIXPathStyle`

`kQTHFSPathStyle`

`kQTWindowsPathStyle`

outPath

A pointer to a `CFStringRef` entity where a reference to the newly created `CFString` will be returned.

Return Value

See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error. Returns `paramErr` if `outPath` is `NIL`.

Discussion

This function creates a new `CFString` that represents the full pathname of the storage location associated with the data handler passed in `dh`.

Version Notes

Introduced in QuickTime 6.4.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`Movies.h`

QTGetDataHandlerTargetNameCFString

Returns the name of the storage location associated with a data handler.

```
OSErr QTGetDataHandlerTargetNameCFString (
   DataHandler dh,
   CFStringRef *fileName
);
```

Parameters

dh

A data handler component instance that is associated with a file.

fileName

A pointer to a `CFStringRef` entity where a reference to the newly created `CFString` will be returned.

Return Value

See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error. Returns `paramErr` if fileName is `NIL`.

Discussion

This function creates a new `CFString` that represents the name of the storage location associated with the data handler passed in `dh`.

Version Notes

Introduced in QuickTime 6.4.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`Movies.h`

QTGetDataReferenceDirectoryDataReference

Returns a new data reference for a parent directory.

```
OSErr QTGetDataReferenceDirectoryDataReference (
   Handle dataRef,
   OSType dataRefType,
   UInt32 flags,
   Handle *outDataRef,
   OSType *outDataRefType
);
```

Parameters

`dataRef`
An alias data reference to which you want a new data reference that points to the directory.

`dataRefType`
The type the input data reference; must be `AliasDataHandlerSubType`.

`flags`
Currently not used; pass 0.

`outDataRef`
A pointer to a handle in which the newly created alias data reference is returned.

`outDataRefType`
A pointer to memory in which the `OSType` of the newly created data reference is returned.

Return Value
See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error. Returns `paramErr` if either of the output parameters is `NIL`.

Discussion
This function returns a new data reference that points to the parent directory of the storage location specified by the data reference passed in `dataRef`. The new data reference returned will have the same type as `dataRefType`.

Version Notes
Introduced in QuickTime 6.4.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`Movies.h`

QTGetDataReferenceFullPathCFString

Returns the full pathname of the target of the data reference as a CFString.

```
OSErr QTGetDataReferenceFullPathCFString (
   Handle dataRef,
   OSType dataRefType,
   QTPathStyle style,
   CFStringRef *outPath
);
```

Parameters

dataRef
: An alias data reference to which you want a new data reference that points to the directory.

dataRefType
: The type the input data reference; must be `AliasDataHandlerSubType`.

pathStyle
: A constant (see below) that identifies the syntax of the pathname. See these constants:
  `kQTNativeDefaultPathStyle`
  `kQTPOSIXPathStyle`
  `kQTHFSPathStyle`
  `kQTWindowsPathStyle`

outPath
: A pointer to a `CFStringRef` entity where a reference to the newly created `CFString` will be returned.

Return Value
See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error. Returns `paramErr` if either of the output parameters was `NIL` or the value of `dataRefType` is not `AliasDataHandlerSubType`.

Discussion
This function creates a new `CFString` that represents the full pathname of the target pointed to by the input data reference, which must be an alias data reference.

Version Notes
Introduced in QuickTime 6.4.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
CaptureAndCompressIPBMovie
QTExtractAndConvertToMovieFile

Declared In
`Movies.h`

QTGetDataReferenceTargetNameCFString

Returns the name of the target of a data reference as a CFString.

```
OSErr QTGetDataReferenceTargetNameCFString (
   Handle dataRef,
   OSType dataRefType,
   CFStringRef *name
);
```

Parameters

dataRef

An alias data reference to which you want a new data reference that points to its directory.

dataRefType

The type the input data reference; must be `AliasDataHandlerSubType`.

name

A pointer to a `CFStringRef` entity where a reference to the newly created `CFString` will be returned.

Return Value

See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error. Returns `paramErr` if either of the output parameters was `NIL` or the value of `dataRefType` is not `AliasDataHandlerSubType`.

Discussion

This function creates a new `CFString` that represents the name of the target pointed to by the input data reference, which must be an alias data reference.

Version Notes

Introduced in QuickTime 6.4.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`Movies.h`

QTGetDataRefMaxFileOffset

Undocumented

```
OSErr QTGetDataRefMaxFileOffset (
   Movie movieH,
   OSType dataRefType,
   Handle dataRef,
   long *offset
);
```

Parameters

movieH

Undocumented

dataRefType

The type of data reference; see `Data References`. If the data reference is an alias, you must set this parameter to `rAliasType`. See *Inside Macintosh: Files* for more information about aliases and the Alias Manager.

dataRef

A handle to a data reference. The type of information stored in the handle depends upon the `data` reference type specified by the `dataRefType` parameter.

offset

Undocumented

Return Value

You can access Movie Toolbox error returns through GetMoviesError (page 221) and GetMoviesStickyError (page 222), as well as in the function result. See Error Codes.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

Movies.h

QTGetEffectsList

Returns a QT atom container holding a list of the currently installed effects components.

```
OSErr QTGetEffectsList (
   QTAtomContainer *returnedList,
   long minSources,
   long maxSources,
   QTEffectListOptions getOptions
);
```

Parameters

returnedList

If the function returns noErr, this parameter contains a newly created QT atom container holding a list of their currently installed effects. Any data stored in the parameter on entry is overwritten by the list of effects. It is the responsibility of the calling application to dispose of the storage by calling QTDisposeAtomContainer (page 1427) once the list is no longer required.

minSources

The minimum number of sources that an effect must have to be added to the list. Pass -1 as this parameter to specify no minimum.

maxSources

The maximum number of sources that an effect can have to be added to the list. Pass -1 as this parameter to specify no maximum. The minSources and maxSources parameters allow you to restrict which effects are returned in the list, by specifying the minimum and maximum number of sources that qualifying effects can have.

getOptions

Options (see below) that control which effects are added to the list. If you pass 0, the function includes every effect, except the "none" effect and any prohibited by the values of minSources and maxSources. See these constants:

elOptionsIncludeNoneInList

Return Value

You can access Movie Toolbox error returns through GetMoviesError (page 221) and GetMoviesStickyError (page 222), as well as in the function result. See Error Codes.

Discussion
The returned list contains two atoms for each effect component. The first atom, of type `kEffectNameAtom`, contains the name of the effect. The second atom, of type `kEffectTypeAtom`, contains the type of the effect, which is the sub-type of the effect component. This list is sorted alphabetically on the names of the effects. You can constrain the `list` to certain types of effects, such as those that take two sources. Use this function to obtain a list of effects that you can pass to `QTCreateStandardParameterDialog` (page 1424).

Special Considerations

This function can take a fairly long time to execute, as it searches the system for installed effects components. You will normally want to call this function once when your application starts, or after a pair of suspend and resume events.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
makeeffectslideshow

qteffects

qteffects.win

qtshoweffect

samplemakeeffectmovie.win

Declared In
`Movies.h`

QTGetEffectsListExtended

Provides for more advanced filtering of effects to be placed into the effect list.

```
OSErr QTGetEffectsListExtended (
   QTAtomContainer *returnedList,
   long minSources,
   long maxSources,
   QTEffectListOptions getOptions,
   OSType majorClass,
   OSType minorClass,
   QTEffectListFilterUPP filterProc,
   void *filterRefCon
);
```

Parameters

returnedList

A pointer to an atom container in which the effects list is returned.

minSources

The minimum number of sources that an effect must have to be added to the list. Pass -1 to specify no minimum.

maxSources

The maximum number of sources that an effect can have to be added to the list. Pass -1 to specify no maximum.

getOptions
: The options for populating the list.

majorClass
: The major class to include, or 0 for all.

minorClass
: The minor class to include, or 0 for all.

filterProc
: A `QTEffectListFilterProc` callback that you can use for additional client filtering. The callback is called for each effect that passes the other criteria for inclusion. If it returns TRUE, the effect is included in the list. Note that your callback may receive multiple effects from various manufacturers. If you return TRUE for multiple effects of a given type, only the one with the higher parameter version number will be included. If you wish to filter for other criteria, such as for a given manufacturer, you can return FALSE for rejected effects and TRUE for those that you prefer.

filterRefCon
: 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

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

This routine provides for more advanced filtering of effects to be placed into the effect list. The `minSources` and `maxSources` parameters allow you to restrict which effects are returned in the list, by specifying the minimum and maximum number of sources that qualifying effects can have. Applications can filter on the number of input sources or on an effect's major or minor class. They can also achieve custom filtering through a callback.

Version Notes

Introduced in QuickTime 6.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`Movies.h`

QTGetEffectSpeed

Returns the speed of the effect, expressed in frames per second.

```
OSErr QTGetEffectSpeed (
   QTAtomContainer parameters,
   Fixed *pFPS
);
```

Parameters

parameters
: Contains parameter values for the effect.

pFPS

The speed of the effect is returned in this parameter, expressed in frames per second. Effects can also return the pre-defined constant `effectIsRealtime` (see below) as their speed. See these constants:

`effectIsRealtime`

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

The value returned should not be treated as an absolute measurement of effect performance. In particular, most effects only return one value, regardless of parameter settings and hardware. This value is an estimate of execution speed on a reference hardware platform. Actual performance will vary depending on hardware, configuration and parameter options.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Fiendishthngs

Declared In

`Movies.h`

QTGetMovieRestrictions

Returns the restrictions, if any, for a given movie.

```
OSErr QTGetMovieRestrictions (
   Movie theMovie,
   QTRestrictionSet *outRestrictionSet,
   UInt32 *outSeed
);
```

Parameters

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

outRestrictionSet

A pointer to a `QTRestrictionSetRecord` structure. If there are no restrictions, this parameter returns `NIL`. See `Movie Restrictions`.

outSeed

A pointer to a long integer. Each change to the restriction set will change this value. You can use this value to detect alterations of the restriction set.

Return Value

Returns `qtOperationNotAuthorizedErr` if a restricted operation is attempted. You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

You can use this function to preflight an operation on a movie to determine whether or not to perform the operation. If a restricted operation is attempted, it will fail and the function will return `qtOperationNotAuthorizedErr`.

Version Notes

Introduced in QuickTime 6.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`Movies.h`

QTGetNextChildType

Returns the next atom type in the child list of the specified parent atom.

```
QTAtomType QTGetNextChildType (
   QTAtomContainer container,
   QTAtom parentAtom,
   QTAtomType currentChildType
);
```

Parameters

`container`
: The atom container that contains the parent atom.

`parentAtom`
: The parent atom for this operation.

`currentChildType`
: The last atom type retrieved by this function.

Return Value

The next atom type in the child list of the atom specified by `parentAtom`.

Discussion

You can call this function to iterate through the `atom` types in a parent atom's child list. To retrieve the first atom type, you should set the value of the `currentChildType` parameter to 0. To retrieve subsequent atom types, you should set the value of the `currentChildType` parameter to the `atom` type retrieved by the previous call to this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

QTGetSupportedRestrictions

Reports the movie restrictions enforced by the currently running version of QuickTime.

```
OSErr QTGetSupportedRestrictions (
   OSType inRestrictionClass,
   UInt32 *outRestrictionIDs
);
```

Parameters

inRestrictionClass

Specifies the class of restrictions to be reported: `kQTRestrictionClassSave` or `kQTRestrictionClassEdit`. See `Movie Restrictions`.

outRestrictionIDs

A pointer to the restrictions in force for the class passed in `inRestrictionClass`. See `Movie Restrictions`.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 6.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`Movies.h`

QTInsertChild

Creates a new child atom of the specified parent atom.

```
OSErr QTInsertChild (
   QTAtomContainer container,
   QTAtom parentAtom,
   QTAtomType atomType,
   QTAtomID id,
   short index,
   long dataSize,
   void *data,
   QTAtom *newAtom
);
```

Parameters

container

The atom container that contains the parent atom. The atom container must not be locked.

parentAtom

The parent atom within the atom container.

atomType

The type of the new atom to be inserted.

id

The ID of the new atom to be inserted. This ID must be unique among atoms of the same type for the specified parent. If you set this parameter to 0, the function assigns a unique ID to the atom.

index

The index of the new atom among atoms with the same parent. To insert the first atom for the specified parent, you should set this parameter to 1. To insert an atom as the last atom in the child list, you should set this parameter to 0. Index values greater than the index of the last atom in the child list plus 1 are invalid.

dataSize

The size of the data for the new atom. If the new atom is to be a parent atom or if you want to add the atom's data later, you should pass 0 for this parameter. To create the new atom as a leaf atom that contains data, you should specify the data using the `data` parameter and and its size using the `dataSize` parameter.

data

A pointer to a buffer containing the data for the new atom. If you set the value of the `dataSize` parameter to 0, you should pass `NIL` for this parameter.

newAtom

A pointer to data of type `QTAtom`. On return, this parameter points to the newly created atom. You can pass `NIL` for this parameter if you don't need a reference to the newly created atom.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

You call this function to create a new child atom. The new child atom has the specified atom type and atom ID, and is inserted into its parent atom's child list at the specified index. Any existing atoms at the same index or greater are moved toward the end of the child list.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qtwiredactions

qtwiredactions.win

qtwiredsprites

qtwiredspritesjr

qtwiredspritesjr.win

Declared In

`Movies.h`

QTInsertChildren

Inserts a container of atoms as children of the specified parent atom.

```
OSErr QTInsertChildren (
   QTAtomContainer container,
   QTAtom parentAtom,
   QTAtomContainer childrenContainer
);
```

Parameters

container
The atom container that contains the parent atom. The atom container must not be locked.

parentAtom
The parent atom within the atom container.

childrenContainer
The atom container that contains the child atoms to be inserted.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
You call this function to insert a container of atoms as children of a parent atom in another atom container. Each child atom is inserted as the last atom of its type and is assigned a corresponding index. The ID of a child atom to be inserted must not duplicate that of an existing child atom of the same type. The following code shows how you can use this function to create a container, insert an atom, and insert another container as a child of the atom:

```
// QTInsertChildren coding example
FailOSErr (QTInsertChild (outerContainer, kParentAtomIsContainer,
    kSpriteAtomType, spriteID, 0, 0, NIL, &newParentAtom));
FailOSErr (QTInsertChildren (outerContainer, newParentAtom,
    innerContainer));
```

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qteffects.win
qtwiredactions
qtwiredactions.win
qtwiredsprites
qtwiredspritesjr

Declared In
`Movies.h`

QTIsStandardParameterDialogEvent

Determines if a Macintosh event is processed by a standard parameter dialog box created by QTCreateStandardParameterDialog.

```
OSErr QTIsStandardParameterDialogEvent (
   EventRecord *pEvent,
   QTParameterDialog createdDialog
);
```

Parameters

`pEvent`
    The Macintosh event.

`createdDialog`
    The reference to the standard parameters dialog box that is returned by `QTCreateStandardParameterDialog` (page 1424).

Return Value
See below.

Discussion
After you create a standard parameter dialog box, pass every Macintosh event through this function to determine if your application should handle the event. Once the dialog box has been confirmed or cancelled by the user, you should no longer call this function.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
MakeEffectMovie
makeeffectslideshow
qteffects.win
QTEffectsDialog - Cocoa
samplemakeeffectmovie.win

Declared In
`Movies.h`

QTLockContainer

Locks an atom container in memory.

```
OSErr QTLockContainer (
   QTAtomContainer container
);
```

Parameters

`container`
    The atom container to be locked.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
You should call this function to lock an atom container before calling `QTGetAtomDataPtr` (page 1433) to directly access a leaf atom's data. When you have finished accessing a leaf atom's data, you should call `QTUnlockContainer` (page 1469). You may make nested pairs of calls to `QTLockContainer` and `QTUnlockContainer`; you don't need to check the current state of the container first.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
makeeffectslideshow

qteffects.win

qtmusic.win

samplemakeeffectmovie.win

vrbackbuffer.win

Declared In
`Movies.h`

QTMovieNeedsTimeTable

Returns whether a movie is being progressively downloaded.

```
OSErr QTMovieNeedsTimeTable (
   Movie theMovie,
   Boolean *needsTimeTable
);
```

Parameters

theMovie
: The movie for this operation. Your application obtains this identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

needsTimeTable
: If TRUE, the movie is being progressively downloaded. If an error occurs, this parameter is set to FALSE.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
A movie can be progressively downloaded when its data is received over a network connection or other slow data channel. Progressive downloads are not necessary when the data for the movie is on a local disk. The Movie Toolbox creates a time table for a movie when either this function or `GetMaxLoadedTimeInMovie` (page 1342) is called for the movie, but the time table is used only by the toolbox and is not accessible to applications. The toolbox disposes of the time table when the download is complete.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTCarbonShell

Declared In
`Movies.h`

QTNewAlias

Creates a Mac OS alias to a file.

```
OSErr QTNewAlias (
   const FSSpec *fss,
   AliasHandle *alias,
   Boolean minimal
);
```

Parameters

`fss`

A pointer to an `FSSpec` structure that specifies a file.

`alias`

On return, a pointer to a handle to a new `AliasRecord` structure that defines an alias to the file. If the function was unable to create an alias, the handle is set to `NIL`. This function does not create relative aliases. For further information about Mac OS file aliases, see Chapter 4 of *Inside Macintosh: Files*.

`minimal`

If you pass TRUE, the function writes in the `AliasRecord` structure only the target name, parent directory ID, volume name and creation date, and volume mounting information. If you pass FALSE, it fills out the structure fully.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qtcapture

qtcapture.win

qtdataref

ThreadsImporter

ThreadsImportMovie

Declared In
`Movies.h`

QTNewAtomContainer

Creates a new atom container.

```
OSErr QTNewAtomContainer (
   QTAtomContainer *atomData
);
```

Parameters

atomData

A pointer to an unallocated atom container data structure. On return, this parameter points to an allocated atom container.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

This function creates a new, empty atom container structure. Once you have created an atom container, you can manipulate it using the atom container functions. The following example illustrates using this function to create a new QT atom container and add an atom:

```
// QTNewAtomContainer coding example
QTAtom firstAtom;
QTAtomContainer container;
OSErr err
err =QTNewAtomContainer (&container);
if (!err)
    err =QTInsertChild (container, kParentAtomIsContainer, 'abcd',
        1000, 1, 0, NIL, &firstAtom);
```

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qtactiontargets

qtactiontargets.win

qteffects.win

qtspritesplus.win

qtwiredspritesjr

Declared In

`Movies.h`

QTNewDataReferenceFromCFURL

Creates a URL data reference from a CFURL.

```
OSErr QTNewDataReferenceFromCFURL (
   CFURLRef url,
   UInt32 flags,
   Handle *outDataRef,
   OSType *outDataRefType
);
```

Parameters

url

A reference to a Core Foundation struct that represents the URL to which you want a URL data reference. These structs contain two parts: the string and a base URL, which may be empty. With a relative URL, the string alone does not fully specify the address; with an absolute URL it does.

flags

Currently not used; pass 0.

outDataRef

A pointer to a handle in which the newly created alias data reference is returned.

outDataRefType

A pointer to memory in which the `OSType` of the newly created data reference is returned.

Return Value

See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error. Returns `paramErr` if either of the output parameters is `NIL`

Discussion

The new URL data reference returned can be passed to other Movie Toolbox calls that take a data reference.

Version Notes

Introduced in QuickTime 6.4.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

CIVideoDemoGL

ComboBoxPrefs

SimpleAudioExtraction

Declared In

`Movies.h`

QTNewDataReferenceFromFSRef

Creates an alias data reference from a file specification.

```
OSErr QTNewDataReferenceFromFSRef (
   const FSRef *fileRef,
   UInt32 flags,
   Handle *outDataRef,
   OSType *outDataRefType
);
```

Parameters

fileRef
: A pointer to an opaque file system reference.

flags
: Currently not used; pass 0.

outDataRef
: A pointer to a handle in which the newly created alias data reference is returned.

outDataRefType
: A pointer to memory in which the `OSType` of the newly created data reference is returned.

Return Value

See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error. Returns `paramErr` if either of the output parameters is `NIL`

Discussion

You can use File Manager functions to construct a file specification for a file to which you want the new alias data reference to point. Then you can pass the reference to other Movie Toolbox functions that take a data reference. To construct a file specification, the file must already exist. To create an alias data reference for a file that does not exist yet, such as a new file to be created by a Movie Toolbox function, call `QTNewDataReferenceFromFSRefCFString`.

Version Notes

Introduced in QuickTime 6.4.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

BackgroundExporter

QTCarbonCoreImage101

QTCarbonShell

QTMetaData

ThreadsExportMovie

Declared In

`Movies.h`

QTNewDataReferenceFromFSRefCFString

Creates an alias data reference from a file reference pointing to a directory and a file name.

```
OSErr QTNewDataReferenceFromFSRefCFString (
   const FSRef *directoryRef,
   CFStringRef fileName,
   UInt32 flags,
   Handle *outDataRef,
   OSType *outDataRefType
);
```

Parameters

directoryRef
A pointer to an opaque file specification that specifies the directory of the newly created alias data reference.

fileName
A reference to a `CFString` that specifies the name of the file.

flags
Currently not used; pass 0.

outDataRef
A pointer to a handle in which the newly created alias data reference is returned.

outDataRefType
A pointer to memory in which the `OSType` of the newly created data reference is returned.

Return Value
See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error. Returns `paramErr` if either of the output parameters is `NIL`

Discussion
This function is useful for creating an alias data reference to a file that does not exist yet. Note that you cannot construct an `FSRef` for a nonexistent file. You can use File Manager functions to construct an `FSRef` for the directory. Depending on where your file name comes from, you may already have it in a form of `CFString`, or you may have to call `CFString` functions to create a new `CFString` for the file name. Then you can pass the new alias data reference to other Movie Toolbox functions that take a data reference. If you already have an `FSRef` for the file you want, you can call `QTNewDataReferenceFromFSRef` instead.

Version Notes
Introduced in QuickTime 6.4.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
CaptureAndCompressIPBMovie
QTExtractAndConvertToMovieFile

Declared In
`Movies.h`

QTNewDataReferenceFromFSSpec

Creates an alias data reference from a file specification of type FSSpec.

```
OSErr QTNewDataReferenceFromFSSpec (
   const FSSpec *fsspec,
   UInt32 flags,
   Handle *outDataRef,
   OSType *outDataRefType
);
```

Parameters

fsspec

A pointer to an opaque file system reference.

flags

Currently not used; pass 0.

outDataRef

A pointer to a handle in which the newly created alias data reference is returned.

outDataRefType

A pointer to memory in which the `OSType` of the newly created data reference is returned.

Return Value

See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error. Returns `paramErr` if either of the output parameters is `NIL`

Discussion

You can use File Manager functions to construct an `FSSpec` structure to specify a file. Then you can pass the new alias data reference to other Movie Toolbox functions that take a data reference. Because of the limitations of its data structure, an `FSSpec` may not work for a file with long or Unicode file names. Generally, you should use either `QTNewDataReferenceFromFSRef` or `QTNewDataReferenceFromFSRefCFString` instead.

Version Notes

Introduced in QuickTime 6.4.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`Movies.h`

QTNewDataReferenceFromFullPathCFString

Creates an alias data reference from a CFString that represents the full pathname of a file.

```
OSErr QTNewDataReferenceFromFullPathCFString (
   CFStringRef filePath,
   QTPathStyle pathStyle,
   UInt32 flags,
   Handle *outDataRef,
   OSType *outDataRefType
);
```

Parameters

filePath

A `CFString` that represents the full pathname of a file.

pathStyle
A constant (see below) that identifies the syntax of the pathname. See these constants:
`kQTNativeDefaultPathStyle`
`kQTPOSIXPathStyle`
`kQTHFSPathStyle`
`kQTWindowsPathStyle`

flags
Currently not used; pass 0.

outDataRef
A pointer to a handle in which the newly created alias data reference is returned.

outDataRefType
A pointer to memory in which the `OSType` of the newly created data reference is returned.

Return Value

See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error. Returns `paramErr` if either of the output parameters is `NIL`

Discussion

You need to specify the syntax of the pathname as one of the `QTPathStyle` constants. The new alias data reference created can be passed to other Movie Toolbox calls that take a data reference.

Version Notes

Introduced in QuickTime 6.4.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

ASCIIMoviePlayerSample
Fiendishthngs
Quartz Composer QCTV
SCAudioCompress
WhackedTV

Declared In

`Movies.h`

QTNewDataReferenceFromURLCFString

Creates a URL data reference from a CFString that represents a URL string.

```
OSErr QTNewDataReferenceFromURLCFString (
   CFStringRef urlString,
   UInt32 flags,
   Handle *outDataRef,
   OSType *outDataRefType
);
```

Parameters

urlString
A `CFString` that represents a URL string.

flags

Currently not used; pass 0.

outDataRef

A pointer to a handle in which the newly created alias data reference is returned.

outDataRefType

A pointer to memory in which the `OSType` of the newly created data reference is returned.

Return Value

See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error. Returns `paramErr` if either of the output parameters is `NIL`

Discussion

The new URL data reference returned can be passed to other Movie Toolbox calls that take a data reference.

Version Notes

Introduced in QuickTime 6.4.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

QTCarbonShell

QuickTimeMovieControl

Declared In

`Movies.h`

QTNewDataReferenceWithDirectoryCFString

Creates an alias data reference from another alias data reference pointing to the parent directory and a CFString that contains the file name.

```
OSErr QTNewDataReferenceWithDirectoryCFString (
   Handle inDataRef,
   OSType inDataRefType,
   CFStringRef targetName,
   UInt32 flags,
   Handle *outDataRef,
   OSType *outDataRefType
);
```

Parameters

inDataRef

An alias data reference pointing to the parent directory.

inDataRefType

The type of the parent directory data reference; it must be `AliasDataHandlerSubType`.

targetName

A reference to a `CFString` containing the file name.

flags

Currently not used; pass 0.

outDataRef

A pointer to a handle in which the newly created alias data reference is returned.

outDataRefType

A pointer to memory in which the `OSType` of the newly created data reference is returned.

Return Value

See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error.

Discussion

In conjunction with `QTGetDataReferenceDirectoryDataReference`, this function is useful to construct an alias data reference to a file in the same directory as the one you already have a data reference for. Then you can pass the new alias data reference to other Movie Toolbox functions that take a data reference.

Version Notes

Introduced in QuickTime 6.4.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`Movies.h`

QTNewTween

Undocumented

```
OSErr QTNewTween (
   QTTweener *tween,
   QTAtomContainer container,
   QTAtom tweenAtom,
   TimeValue maxTime
);
```

Parameters

tween

A pointer to a pointer to a `QTTweenerRecord` structure.

container

Undocumented

tweenAtom

Undocumented

maxTime

Undocumented

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Dimmer2Effect

Dimmer2Effect.win

Declared In
`Movies.h`

QTNextChildAnyType

Returns the next atom in the child list of the specified parent atom.

```
OSErr QTNextChildAnyType (
   QTAtomContainer container,
   QTAtom parentAtom,
   QTAtom currentChild,
   QTAtom *nextChild
);
```

Parameters

`container`
The atom container that contains the parent atom.

`parentAtom`
The parent atom for this operation.

`currentChild`
The last atom retrieved by this function. To retrieve the first atom in the child list, set the value of `currentChild` to 0.

`nextChild`
A pointer to an uninitialized QT atom data structure. On return, the data structure contains the offset of the next atom in the child list after the atom specified by `currentChild`, or 0 if the atom specified by `currentChild` was the last atom in the list.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
You can call this function to iterate through all the atoms in a parent atom's child list, regardless of their types and IDs.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
addflashactions
addflashactions.win
Fiendishthngs
SimpleVideoOut

Declared In
`Movies.h`

QTRegisterAccessKey

Registers an access key.

```
OSErr QTRegisterAccessKey (
   Str255 accessKeyType,
   long flags,
   Handle accessKey
);
```

Parameters

accessKeyType

The access key type of the key to be registered.

flags

Flags that specify the operation of this function. To register a system access key, set the `kAccessKeySystemFlag` flag (see below). To register an application access key, set this parameter to 0. See these constants:

`kAccessKeySystemFlag`

accessKey

A handle to the key to be registered.

Return Value

See `Error Codes`. Returns `noErr` if there is no error or if the access key has already been registered.

Discussion

Most access keys are strings. A string stored in the `accessKey` handle does not include a trailing zero or leading length byte; to get the length of the string, get the size of the handle. If the access key has already been registered, no error is returned, and the request is simply ignored.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

QTRemoveAtom

Removes an atom and its children from the specified atom container.

```
OSErr QTRemoveAtom (
   QTAtomContainer container,
   QTAtom atom
);
```

Parameters

container

The atom container for this operation. The atom container must not be locked.

atom

The atom to be removed from the container.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

You call this function to remove a particular atom and its children from an atom container. To remove all the atoms in an atom container, you should use `QTRemoveChildren` (page 1462).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

addvractions

addvractions.win

Declared In

`Movies.h`

QTRemoveChildren

Removes all the children of an atom from the specified atom container.

```
OSErr QTRemoveChildren (
   QTAtomContainer container,
   QTAtom atom
);
```

Parameters

`container`

The atom container for this operation. The atom container must not be locked.

`atom`

The atom whose children should be removed. To remove all the atoms in the atom container, pass a value of `kParentAtomIsContainer`.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

MovieSprites

qtsprites

qtsprites.win

qtwiredsprites

WiredSprites

Declared In

`Movies.h`

QTReplaceAtom

Replaces the contents of an atom and its children with a different atom and its children.

```
OSErr QTReplaceAtom (
   QTAtomContainer targetContainer,
   QTAtom targetAtom,
   QTAtomContainer replacementContainer,
   QTAtom replacementAtom
);
```

Parameters

targetContainer
: The atom container that contains the atom to be replaced. The atom container must not be locked.

targetAtom
: The atom to be replaced.

replacementContainer
: The atom container that contains the replacement atom.

replacementAtom
: The replacement atom.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

The target atom and the replacement atom must be of the same type. The target atom maintains its original atom ID. This function does not modify the replacement container.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

addvractions

addvractions.win

Declared In

`Movies.h`

QTRestrictionsGetIndClass

Reports the class of a movie restriction.

```
OSErr QTRestrictionsGetIndClass (
   QTRestrictionSet inRestrictionSet,
   long inIndex,
   OSType *outClass
);
```

Parameters

inRestrictionSet
A pointer to a `QTRestrictionSetRecord` structure containing the set of restrictions to be reported.

inIndex
The index of a restriction.

outClass
A pointer to the class of restrictions of `inIndex`: `kQTRestrictionClassSave` or `kQTRestrictionClassEdit`. See `Movie Restrictions`.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 6.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`Movies.h`

QTRestrictionsGetInfo

Reports information about the restrictions in a specified restriction set.

```
OSErr QTRestrictionsGetInfo (
   QTRestrictionSet inRestrictionSet,
   long *outRestrictionClassCount,
   long *outSeed
);
```

Parameters

inRestrictionSet
A pointer to a `QTRestrictionSetRecord` structure containing the set of restrictions to be reported.

outRestrictionClassCount
The number of restrictions classes currently in the restriction set.

outSeed
A pointer to a long integer. Each alteration of the restriction set will change this value.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

If you want to determine all the restrictions, use this routine to get their count.

Version Notes
Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`Movies.h`

QTRestrictionsGetItem

Retrieves specific movie restrictions.

```
OSErr QTRestrictionsGetItem (
   QTRestrictionSet inRestrictionSet,
   OSType inRestrictionClass,
   UInt32 *outRestrictions
);
```

Parameters

inRestrictionSet
A pointer to a `QTRestrictionSetRecord` structure containing the set of restrictions for a given movie.

inRestrictionClass
Specifies the class of restrictions to be reported: `kQTRestrictionClassSave` or `kQTRestrictionClassEdit`. See `Movie Restrictions`.

outRestrictions
A pointer to a long integer holding constants that indicate individual restrictions. See `Movie Restrictions`.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
If the movie has no restrictions, `outRestrictions` returns 0. If a restriction class is not available, the function won't return an error but `outRestrictions` will be set to 0.

Version Notes
Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`Movies.h`

QTSetAtomData

Changes the data of a leaf atom.

```
OSErr QTSetAtomData (
   QTAtomContainer container,
   QTAtom atom,
   long dataSize,
   void *atomData
);
```

Parameters

container
: The atom container that contains the atom to be modified.

atom
: The atom to be modified.

dataSize
: The length, in bytes, of the data pointed to by the `atomData` parameter.

atomData
: A pointer to the new data for the atom.

Return Value

Only leaf atoms contain data; this function returns an error if you pass it to a nonleaf atom. You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

You call this function to replace a leaf atom's data with new data. The atom container specified by the `container` parameter should not be locked. The following code illustrates using this function to update an atom container that describes a sprite:

```
// QTSetAtomData coding example
OSErr SetSpriteData (QTAtomContainer sprite, Point *location,
    short *visible, short *layer, short *imageIndex)
{
    OSErr err =noErr;
    QTAtom propertyAtom;

    // if the sprite's visible property has a new value
    if (visible)
    {
        // retrieve the atom for the visible property
        // -- if none exists, insert one
        if ((propertyAtom =QTFindChildByIndex (sprite,
            kParentAtomIsContainer, kSpritePropertyVisible, 1,
            NIL)) ==0)
            FailOSErr (QTInsertChild (sprite, kParentAtomIsContainer,
                kSpritePropertyVisible, 1, 1, sizeof(short), visible,
                NIL))

        // if an atom does exist, update its data
        else
            FailOSErr (QTSetAtomData (sprite, propertyAtom,
                sizeof(short), visible));
    }
```

Special Considerations

This function may move memory; if the pointer specified by the `atomData` parameter is a dereferenced handle, you should lock the handle.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qteffects.win
qtwiredsprites
qtwiredsprites.win
qtwiredspritesjr
qtwiredspritesjr.win

Declared In
`Movies.h`

QTSetAtomID

Changes the ID of an atom.

```
OSErr QTSetAtomID (
   QTAtomContainer container,
   QTAtom atom,
   QTAtomID newID
);
```

Parameters

container
The atom container for this operation.

atom
The atom to be modified. You cannot change the ID of the container by passing 0 for the `atom` parameter.

newID
The new ID for the atom. You cannot change an atom's ID to an ID already assigned to a sibling atom of the same type.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

QTStandardParameterDialogDoAction

Lets you change some of the default behaviors of the standard parameter dialog box.

```
OSErr QTStandardParameterDialogDoAction (
   QTParameterDialog createdDialog,
   long action,
   void *params
);
```

Parameters

createdDialog

The reference to the dialog box created by calling `QTCreateStandardParameterDialog` (page 1424).

action

Determines which of the actions (see below) supported by this function will be performed. See these constants:

`pdActionSetAppleMenu`

`pdActionSetEditMenu`

`pdActionSetPreviewPicture`

`pdActionSetDialogTitle`

`pdActionGetSubPanelMenu`

`pdActionActivateSubPanel`

`pdActionConductStopAlert`

params

Optional parameters to the `action`. The type passed in this parameter depends on the value of the `action` parameter.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

This function allows you to change some of the default behaviors of a standard parameter dialog box you create using the `QTCreateStandardParameterDialog` (page 1424) function. To choose which of the available customizations to perform, pass an action selector value in the `action` parameter and, optionally, a single parameter in `params`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

MakeEffectMovie

qteffects

qteffects.win

samplemakeeffectmovie

samplemakeeffectmovie.win

Declared In

`Movies.h`

QTSwapAtoms

Swaps the contents of two atoms in an atom container.

```
OSErr QTSwapAtoms (
   QTAtomContainer container,
   QTAtom atom1,
   QTAtom atom2
);
```

Parameters

container
The atom container for this operation.

atom1
Specifies an atom to be swapped with the atom specified by atom2.

atom2
Specifies an atom to be swapped with the atom specified by atom1.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
After swapping, the ID and index of each atom remains the same. The two atoms specified must be of the same type. Either atom may be a leaf atom or a container atom.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

QTUnlockContainer

Unlocks an atom container in memory.

```
OSErr QTUnlockContainer (
   QTAtomContainer container
);
```

Parameters

container
The atom container to be unlocked.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
You should call this function to unlock an atom container when you have finished accessing a leaf atom's data. You may make nested pairs of calls to `QTLockContainer` (page 1449) and this function; you don't need to check the current state of the container first.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

makeeffectslideshow

qteffects.win

qtmusic.win

samplemakeeffectmovie.win

vrbackbuffer.win

Declared In

`Movies.h`

QTUnregisterAccessKey

Removes a previously registered access key.

```
OSErr QTUnregisterAccessKey (
   Str255 accessKeyType,
   long flags,
   Handle accessKey
);
```

Parameters

accessKeyType

The access key type of the key to be removed.

flags

Flags (see below) that specify the operation of this function. To remove a system access key, set the `kAccessKeySystemFlag` flag. To remove an application access key, set this parameter to 0. See these constants:

`kAccessKeySystemFlag`

accessKey

The key to be removed.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

Most access keys are strings. A string stored in the `accessKey` handle does not include a trailing zero or a leading length byte.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

RemoveMovieExecuteWiredActionsProc

Removes a MovieExecuteWiredActionsProc callback from a movie.

```
OSErr RemoveMovieExecuteWiredActionsProc (
   Movie theMovie,
   MovieExecuteWiredActionsUPP proc,
   void *refCon
);
```

Parameters

theMovie

A movie identifier. Your application obtains this identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

proc

A `MovieExecuteWiredActionsProc` callback that was previously installed using `AddMovieExecuteWiredActionsProc` (page 1298).

refCon

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

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

RemoveMovieResource

Removes a movie resource from a specified movie file.

```
OSErr RemoveMovieResource (
   short resRefNum,
   short resId
);
```

Parameters

resRefNum

Identifies the movie file that contains the movie resource. Your application obtains this value from `OpenMovieFile` (page 1416).

resId

ID of the resource to be removed.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

RemoveSoundDescriptionExtension

Removes an extension from a SoundDescription structure.

```
OSErr RemoveSoundDescriptionExtension (
   SoundDescriptionHandle desc,
   OSType idType
);
```

Parameters

desc
A handle to the `SoundDescription` structure to remove the extension from.

idType
A four-byte signature identifying the type of data being removed from the `SoundDescription` structure.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

RemoveUserData

Removes an item from a user data list.

```
OSErr RemoveUserData (
   UserData theUserData,
   OSType udType,
   long index
);
```

Parameters

theUserData
The user data list for this operation. You obtain this list reference by calling `GetMovieUserData` (page 225), `GetTrackUserData` (page 1617), or `GetMediaUserData` (page 1595).

udType

The item's type value.

index

The item's index value. This parameter must specify an item in the user data list identified by the `theUserData` parameter.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
After the Movie Toolbox removes the item, it renumbers the remaining items of that type so that their index values are sequential and start at 1.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
MakeEffectMovie

qtactiontargets

qtactiontargets.win

qteffects.win

qtgraphics.win

Declared In
`Movies.h`

RemoveUserDataText

Removes language-tagged text from an item in a user data list.

```
OSErr RemoveUserDataText (
   UserData theUserData,
   OSType udType,
   long index,
   short itlRegionTag
);
```

Parameters

theUserData

The user data list for this operation. You obtain this list reference by calling the `GetMovieUserData` (page 225), `GetTrackUserData` (page 1617), or `GetMediaUserData` (page 1595).

udType

The item's type value.

index

The item's index value. This parameter must specify an item in the user data list identified by the `theUserData` parameter.

itlRegionTag

The language code of the text to be removed. See `Localization Codes`.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

SetMediaDataRef

Changes the file that the specified media identifies as the location for its data storage.

```
OSErr SetMediaDataRef (
   Media theMedia,
   short index,
   Handle dataRef,
   OSType dataRefType
);
```

Parameters

`theMedia`
: Specifies The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (page 1630) and `GetTrackMedia` (page 1612). See `Media Identifiers`.

`index`
: A pointer to a short integer. The Movie Toolbox returns the index value that is assigned to the new data reference. Your application can use this index to identify the reference to other Movie Toolbox functions, such as `GetMediaDataRef` (page 1342). As with all data reference functions, the index starts with 1. If the Movie Toolbox cannot add the data reference to the media, it sets the returned index value to 0.

`dataRef`
: The data reference. This parameter contains a handle to the information that identifies the file that contains this media's data. The type of information stored in that handle depends upon the value of the `dataRefType` parameter.

`dataRefType`
: The type of data reference. If the data reference is an alias, you must set this parameter to `rAliasType`.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

Don't call this function unless you have a really good reason. However, if you want to resolve your own missing data references, or you are developing a special-purpose kind of application, this function can be quite useful.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

SetMediaDataRefAttributes

Sets a data reference's attributes.

```
OSErr SetMediaDataRefAttributes (
   Media theMedia,
   short index,
   long dataRefAttributes
);
```

Parameters

theMedia
: Specifies The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (page 1630) and `GetTrackMedia` (page 1612). See `Media Identifiers`.

index
: The index value that corresponds to the data reference. It must be less than or equal to the value that is returned by `GetMediaDataRefCount` (page 1344).

dataRefAttributes
: A flag (see below) that determines whether or not the data reference is the movie default. See these constants:
    `kMovieAnchorDataRefIsDefault`

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

SetMediaPlayHints

Provides information to the Movie Toolbox that can influence playback of a single media.

```
void SetMediaPlayHints (
   Media theMedia,
   long flags,
   long flagsMask
);
```

Parameters

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (page 1630) and `GetTrackMedia` (page 1612). See `Media Identifiers`.

flags

The optimizations that can be used with this media. Each bit in this parameter corresponds to a specific optimization; be sure to set unused flags to 0. See these constants:

`hintsScrubMode`

`hintsUseSoundInterp`

`hintsAllowInterlace`

`hintsAllowBlacklining`

`hintsDontPurge`

`hintsInactive`

`hintsHighQuality`

flagsMask

Indicates which flags in the `flags` parameter are to be considered in this operation. For each bit in the `flags` parameter that you want the Movie Toolbox to consider, you must set the corresponding bit in the `flagsMask` parameter to 1. Set unused flags to 0. This allows you to work with a single optimization without altering the settings of other flags.

Return Value

You can access error returns from this function through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). See `Error Codes`.

Discussion

This function accepts a flag in which you specify optimizations that the Movie Toolbox can use during movie playback. These optimizations apply to only the specified media.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

SetMediaPropertyAtom

Sets the property atom container of a media handler.

```
OSErr SetMediaPropertyAtom (
   Media theMedia,
   QTAtomContainer propertyAtom
);
```

Parameters

theMedia
A reference to the media handler for this operation.

propertyAtom
Specifies a QT atom container that contains the property atoms for the track associated with the media handler.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

You can call this function to set properties for the track associated with the specified media handler. The contents of the QT atom container are defined by the media handler. Here is some sample code that uses this function to define the background color for a sprite track:

```
// SetMediaPropertyAtom coding example
// See "Discovering QuickTime," page 360
if (bWithBackgroundPicture) {
    QTAtomContainer         qtacTrackProperties;
    RGBColor                rgbcBackColor;
    rgbcBackColor.red =EndianU16_NtoB(0x8000);
    rgbcBackColor.green =EndianU16_NtoB(0);
    rgbcBackColor.blue =EndianU16_NtoB0(xffff);
    // create a new atom container for sprite track properties
    QTNewAtomContainer(&qtacTrackProperties);
    // add an atom for the background color property
    QTInsertChild(qtacTrackProperties, 0,
        kSpriteTrackPropertyBackgroundColor, 1, 1, sizeof(RGBColor),
        &rgbcBackColor, NIL);
    // set the sprite track's properties
    nErr =SetMediaPropertyAtom(media, qtacTrackProperties);
    QTDisposeAtomContainer(qtacTrackProperties);
}
```

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

MovieSprites

qteffects.win

qtsprites.win

qtwiredactions

qtwiredactions.win

Declared In

`Movies.h`

SetMovieAnchorDataRef

Sets a movie's anchor data reference and type.

```
OSErr SetMovieAnchorDataRef (
   Movie theMovie,
   Handle dataRef,
   OSType dataRefType
);
```

Parameters

theMovie

A movie identifier. Your application obtains this identifier from such functions as NewMovie (page 259), NewMovieFromFile (page 1398), and NewMovieFromHandle (page 1400).

dataRef

A handle to the data reference. The type of information to be placed in the handle depends upon the data reference type specified by dataRefType.

dataRefType

The type of data reference; see Data References.

Return Value

You can access Movie Toolbox error returns through GetMoviesError (page 221) and GetMoviesStickyError (page 222), as well as in the function result. See Error Codes.

Version Notes

Introduced in QuickTime 4.1.

Availability

Available in Mac OS X v10.0 and later.

Declared In

Movies.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 (page 259), NewMovieFromProperties (page 260), NewMovieFromFile, and NewMovieFromHandle (page 1400).

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 1350).

Availability
Available in Mac OS X v10.3 and later.

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` (page 259), `NewMovieFromProperties` (page 260), `NewMovieFromFile`, and `NewMovieFromHandle` (page 1400).

whatMixToMeter
  The applicable mix of audio channels in the movie; see `Movie Audio Mixes`.

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 1352).

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
Core Animation QuickTime Layer
SillyFrequencyLevels

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` (page 259), `NewMovieFromProperties` (page 260), `NewMovieFromFile`, and `NewMovieFromHandle` (page 1400).

gain

A 32-bit floating-point gain value of 0 or greater. This value is multiplied by the movie's volume. 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 1353).

Availability
Available in Mac OS X v10.3 and later.

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` (page 259), `NewMovieFromProperties` (page 260), `NewMovieFromFile`, and `NewMovieFromHandle` (page 1400).

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`.

Availability

Available in Mac OS X v10.3 and later.

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` (page 259), `NewMovieFromProperties` (page 260), `NewMovieFromFile`, and `NewMovieFromHandle` (page 1400).

whatMixToMeter

The applicable mix of audio channels in the movie; see `Movie Audio Mixes`.

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 1355).

Availability
Available in Mac OS X v10.3 and later.

Declared In
`Movies.h`

SetMovieColorTable

Associates a ColorTable structure with a movie.

```
OSErr SetMovieColorTable (
   Movie theMovie,
   CTabHandle ctab
);
```

Parameters

theMovie
The movie for this operation. Your application obtains this identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

ctab
A handle to the `ColorTable` structure. Set this parameter to `NIL` to remove the movie's `ColorTable` structure.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
The `ColorTable` structure you supply may be used to modify the palette of indexed display devices at playback time. If you are using the movie controller, be sure to set the `mcFlagsUseWindowPalette` flag. If you are not using the movie controller, you should retrieve the movie's `ColorTable` structure, using `GetMovieColorTable` (page 1356), and supply it to the Palette Manager.

Special Considerations

The toolbox makes a copy of the `ColorTable` structure, so it is your responsibility to dispose of the structure when you are done with it. If the movie already has a color table, the toolbox uses the new table to replace the old one.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

SetMovieCoverProcs

Sets the callbacks invoked when a movie is covered or uncovered.

```
void SetMovieCoverProcs (
   Movie theMovie,
   MovieRgnCoverUPP uncoverProc,
   MovieRgnCoverUPP coverProc,
   long refcon
);
```

Parameters

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

uncoverProc

Points to a `MovieRgnCoverProc` callback. This function is called whenever one of your movie's tracks is removed from the screen or resized, revealing a previously hidden screen region. If you want to remove this uncover function, set this parameter to `NIL`. When the `uncoverProc` parameter is `NIL` the function uses the default uncover function, which erases the uncovered area.

coverProc

Points to a `MovieRgnCoverProc` callback. The Movie Toolbox calls this function whenever one of your movies covers a portion of the screen. If you want to remove the cover function, set this parameter to `NIL`. When the `coverProc` parameter is `NIL` the function uses the default cover function, which does nothing.

refcon

Specifies a reference constant. Use this parameter to point to a data structure containing any information your callbacks need.

Return Value

You can access error returns from this function through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). See `Error Codes`.

Discussion

If a movie with semi-transparent tracks has a movie uncover procedure, set with this function, the uncover procedure is called before each frame to fill or erase the background.

Version Notes

Before QuickTime 1.6.1, the Movie Toolbox performed the erase, which limited a cover procedure-aware application's options.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Inside Mac Movie TB Code

vrmovies

vrmovies.win

vrscript

vrscript.win

Declared In

`Movies.h`

SetMovieDefaultDataRef

Sets a movie's default data reference and type.

```
OSErr SetMovieDefaultDataRef (
   Movie theMovie,
   Handle dataRef,
   OSType dataRefType
);
```

Parameters

theMovie
: A movie identifier. Your application obtains this identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

dataRef
: A handle to the `data` reference. The type of information to be placed in the handle depends upon the `data` reference type specified by `dataRefType`.

dataRefType
: The type of data reference; see `Data References`.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
ConvertMovieSndTrack
qtdataref
SoundPlayer
SurfaceVertexProgram
ThreadsImportMovie

Declared In
`Movies.h`

SetMovieLanguage

Specifies a movie's localized language or region code.

```
void SetMovieLanguage (
   Movie theMovie,
   long language
);
```

Parameters

theMovie
: The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

language

The movie's language or region code; see `Localization Codes`.

Return Value

You can access error returns from this function through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). See `Error Codes`.

Discussion

The Movie Toolbox examines the movie's alternate groups and selects and enables appropriate tracks. If the Movie Toolbox cannot find an appropriate track, it does not change the movie's language.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

SetMoviePlayHints

Provides information to the Movie Toolbox that can influence movie playback.

```
void SetMoviePlayHints (
   Movie theMovie,
   long flags,
   long flagsMask
);
```

Parameters

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

flags

The optimizations that can be used with this movie. Each bit in the `flags` parameter corresponds to a specific optimization (see below). Be sure to set unused flags to 0. See these constants:

`hintsScrubMode`

`hintsUseSoundInterp`

`hintsAllowInterlace`

`hintsAllowBlacklining`

`hintsDontPurge`

`hintsInactive`

`hintsHighQuality`

flagsMask

Indicates which flags in the `flags` parameter are to be considered in this operation. For each bit in the `flags` parameter that you want the Movie Toolbox to consider, you must set the corresponding bit in the `flagsMask` parameter to 1. Set unused flags to 0. This allows you to work with a single optimization without altering the settings of other flags.

Return Value

You can access error returns from this function through GetMoviesError (page 221) and GetMoviesStickyError (page 222). See Error Codes.

Discussion

This function accepts a flag in which you specify optimizations that the Movie Toolbox can use during movie playback. These optimizations apply to all of the media structures used by the movie.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

MakeEffectMovie

qteffects.win

vrmakeobject

vrmakeobject.win

vrmakepano

Declared In

`Movies.h`

SetMovieProgressProc

Attaches a progress function to a movie.

```
void SetMovieProgressProc (
   Movie theMovie,
   MovieProgressUPP p,
   long refcon
);
```

Parameters

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as NewMovie (page 259), NewMovieFromFile (page 1398), and NewMovieFromHandle (page 1400).

p

Points to your `MovieProgressProc` callback. To remove a movie's progress function, set this parameter to `NIL`. Set this parameter to -1 for the Movie Toolbox to provide a default progress function.

refcon

Specifies a reference constant. Use this parameter to point to a data structure containing any information your callback needs.

Return Value

You can access error returns from this function through GetMoviesError (page 221) and GetMoviesStickyError (page 222). See Error Codes.

Discussion

The Movie Toolbox calls your function only during long operations. It ensures that your progress function is called regularly, but not too often.

The following Movie Toolbox functions use progress functions: `ConvertFileToMovieFile` (page 1552), `CutMovieSelection` (page 1560), `CopyMovieSelection` (page 1558), `AddMovieSelection` (page 1545), and `InsertMovieSegment` (page 1622).

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qtdataref

soundsnippets

soundsnippets.win

vrmakepano

vrmakepano.win

Declared In
`Movies.h`

SetMoviePropertyAtom

Sets a movie's property atom.

```
OSErr SetMoviePropertyAtom (
   Movie theMovie,
   QTAtomContainer propertyAtom
);
```

Parameters

theMovie

A movie identifier. Your application obtains this identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

propertyAtom

A property atom.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

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.

Availability

Available in Mac OS X v10.3 and later.

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 gray. The setting is not stored in the movie. It is only used until the movie is closed, at which time it is not saved.

Availability
Available in Mac OS X v10.3 and later.

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.

Availability
Available in Mac OS X v10.3 and later.

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.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`Movies.h`

SetPosterBox

Sets a poster's boundary rectangle.

```
void SetPosterBox (
   Movie theMovie,
   const Rect *boxRect
);
```

Parameters

theMovie
: The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

boxRect
: A pointer to a `Rect` structure. The Movie Toolbox sets the poster's boundary rectangle to the coordinates specified in the structure referred to by this parameter.

Return Value
You can access error returns from this function through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). See `Error Codes`.

Discussion
You define the poster's image by specifying a time in the movie, using `SetMoviePosterTime` (page 293). You specify the size and position of the poster image with this function. If you don't specify a boundary rectangle for the poster, the Movie Toolbox uses the movie's matrix when it displays the poster.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

SetQuickTimePreference

Sets a particular preference in the QuickTime preferences.

```
OSErr SetQuickTimePreference (
   OSType preferenceType,
   QTAtomContainer preferenceAtom
);
```

Parameters

preferenceType

The type of preference to set (see below); also see `Atom ID Codes`. See these constants:

`ConnectionSpeedPrefsType`

`BandwidthManagementPrefsType`

preferenceAtom

A QT atom containing the preference information.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

MakeEffectMovie

qteffects.win

qtgraphics.win

qtwiredactions

vrbackbuffer.win

Declared In

`Movies.h`

SetSpriteProperty

Sets the specified property of a sprite.

```
OSErr SetSpriteProperty (
   Sprite theSprite,
   long propertyType,
   void *propertyValue
);
```

Parameters

theSprite

The sprite for this operation.

propertyType

The property you want to modify (see below). See these constants:

`kSpritePropertyMatrix`

`kSpritePropertyImageDescription`

`kSpritePropertyImageDataPtr`

`kSpritePropertyVisible`

`kSpritePropertyLayer`

`kSpritePropertyGraphicsMode`

`kSpritePropertyCanBeHitTested`

propertyValue

The new value of the property. Depending on the `property` type, you set the `propertyValue` parameter to either a pointer to the property value or the property value itself, cast as a `void` pointer.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

You animate a sprite by modifying its properties, using this function. It invalidates the sprite's sprite world as needed. Here is sample code that uses this function to modify a sprite's properties:

```
// SetSpriteProperty coding example
// See "Discovering QuickTime," page 345
#define kNumSprites             4
#define kNumSpaceShipImages     24
Rect            gBounceBox;
Sprite          gSprites[kNumSprites];
Rect            gDestRects[kNumSprites];
Point           gDeltas[kNumSprites];
short           gCurrentImages[kNumSprites];
Handle          gCompressedPictures[kNumSpaceShipImages];
void MyMoveSprites (void)
{
    short           nIndex;
    MatrixRecord    matrix;

    SetIdentityMatrix(&matrix);
    // for each sprite
    for (nIndex =0; nIndex < kNumSprites; nIndex++) {
        // modify the sprite's matrix
        OffsetRect(&gDestRects[nIndex], gDeltas[nIndex].h,
                    gDeltas[nIndex].v);

        if ((gDestRects[nIndex].right >
=gBounceBox.right) ||
            (gDestRects[nIndex].left <=gBounceBox.left))
            gDeltas[nIndex].h =-gDeltas[nIndex].h;

        if ((gDestRects[nIndex].bottom >
=gBounceBox.bottom) ||
            (gDestRects[nIndex].top <=gBounceBox.top))
            gDeltas[nIndex].v =-gDeltas[nIndex].v;

        matrix.matrix[2][0] =((long)gDestRects[nIndex].left << 16);
        matrix.matrix[2][1] =((long)gDestRects[nIndex].top << 16);
        SetSpriteProperty(gSprites[nIndex], kSpritePropertyMatrix,
                            &matrix);

        // change the sprite's image
        gCurrentImages[nIndex]++;
        if (gCurrentImages[nIndex] >
=(kNumSpaceShipImages *
                                        (nIndex+1)))
            gCurrentImages[nIndex] =0;
        SetSpriteProperty(gSprites[nIndex], kSpritePropertyImageDataPtr,
            *gCompressedPictures[gCurrentImages[nIndex] / (nIndex+1)]);
    }
}
```

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Desktop Sprites

DesktopSprites

DesktopSprites.win

Declared In
`Movies.h`

SetSpriteWorldClip

Sets a sprite world's clip shape to the specified region.

```
OSErr SetSpriteWorldClip (
   SpriteWorld theSpriteWorld,
   RgnHandle clipRgn
);
```

Parameters
theSpriteWorld
: The sprite world for this operation.

clipRgn
: The new clip shape for the sprite world. The clip shape should be specified in the sprite world's source space, the coordinate system of the sprite layer's graphics world before the sprite world's matrix is applied to it. You may pass a value of `NIL` for this parameter to indicate that there is no longer a clip shape for the sprite world. This means that the whole area is drawn.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
You call this function to change the clip shape of a sprite world. The specified region is owned by the caller and is not copied by this function.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

SetSpriteWorldFlags

Sets flags that govern the behavior of a sprite world.

```
OSErr SetSpriteWorldFlags (
   SpriteWorld spriteWorld,
   long flags,
   long flagsMask
);
```

Parameters

spriteWorld
The sprite world for this operation.

flags
Constants (see below) that govern sprite world behavior. See these constants:
`kScaleSpritesToScaleWorld`
`kSpriteWorldHighQuality`
`kSpriteWorldDontAutoInvalidate`
`kSpriteWorldInvisible`

flagsMask
Indicates which flags in the `flags` parameter are to be considered in this operation. For each bit in the `flags` parameter that you want the Movie Toolbox to consider, set the corresponding bit in the `flagsMask` parameter to 1. Set unused flags to 0. This allows you to work with a single optimization without altering the settings of other flags.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

SetSpriteWorldGraphicsMode

Sets the graphics transfer mode for a sprite world.

```
OSErr SetSpriteWorldGraphicsMode (
   SpriteWorld theSpriteWorld,
   long mode,
   const RGBColor *opColor
);
```

Parameters

theSpriteWorld

The sprite world for this operation.

mode

A long integer; see `Graphics Transfer Modes`.

opColor

A pointer to an `RGBColor` structure. This is the blend value for blends and the transparent color for transparent operations. The toolbox supplies this value to QuickDraw when you draw in `addPin`, `subPin`, `blend`, `transparent`, or `graphicsModeStraightAlphaBlend` mode.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

SetSpriteWorldMatrix

Sets a sprite world's matrix to the specified matrix.

```
OSErr SetSpriteWorldMatrix (
   SpriteWorld theSpriteWorld,
   const MatrixRecord *matrix
);
```

Parameters

theSpriteWorld

The sprite world for this operation.

matrix

A pointer to the new matrix for the sprite world. Transformations may include translation, scaling, rotation, skewing, and perspective. You may pass a value of `NIL` to set the sprite world's matrix to an identity matrix.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

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` (page 1628) and `GetMovieTrack` (page 1601).

`gain`
: A 32-bit floating-point gain value of 0 or greater. This value is multiplied by the track's volume. 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 1370).

Availability
Available in Mac OS X v10.3 and later.

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` (page 1628) and `GetMovieTrack` (page 1601).

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 1371).

Availability
Available in Mac OS X v10.3 and later.

Declared In
`Movies.h`

SetTrackLoadSettings

Specifies a portion of a track that is to be loaded into memory whenever it is played.

```
void SetTrackLoadSettings (
   Track theTrack,
   TimeValue preloadTime,
   TimeValue preloadDuration,
   long preloadFlags,
   long defaultHints
);
```

Parameters

theTrack
: The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

preloadTime
: The starting point of the portion of the track to be preloaded. Set this parameter to -1 if you want to preload the entire track (in this case the function ignores the `preloadDuration` parameter). This parameter should be specified using the movie's time scale.

preloadDuration
: The amount of the track to be preloaded, starting from the time specified in the `preloadTime` parameter. If you are preloading the entire track, the function ignores this parameter.

preloadFlags

Controls when the toolbox preloads the track. The function supports the following flag values: See these constants:

`preloadAlways`

`preloadOnlyIfEnabled`

defaultHints

Specifies playback hints for the track. You may specify any of the supported hints flags.

Return Value

You can access error returns from this function through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). See `Error Codes`.

Discussion

This function allows you to control how the toolbox preloads the tracks in your movie. By using its settings, you make this information part of the movie, so that the preloading takes place every time the movie is opened, without an application having to call `LoadTrackIntoRam` (page 247). Consequently, you should use this feature carefully, so that your movies don't consume large amounts of memory when opened.

Special Considerations

The toolbox transfers this preload information when you call `CopyTrackSettings` (page 1559). In addition, the preload information is preserved when you save or flatten a movie. In flattened movies, the tracks that are to be preloaded are stored at the start of the movie, rather than being interleaved with the rest of the movie data. This improves preload performance.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

SetUserDataItem

Sets an item in a user data list.

```
OSErr SetUserDataItem (
   UserData theUserData,
   void *data,
   long size,
   OSType udType,
   long index
);
```

Parameters

theUserData

The user data list for this operation. You obtain this item reference by calling `GetMovieUserData` (page 225), `GetTrackUserData` (page 1617), or `GetMediaUserData` (page 1595).

data

A pointer to the data item to be set in a user data list.

size

The size of the information pointed to by the `data` parameter.

udType

The type value assigned to the new item.

index

The item's index value. This parameter must specify an item in the user data list identified by `theUserData`. An index value of 0 or 1 implies the first item, which is created if it doesn't already exist.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

MakeEffectMovie

qteffects.win

qtwiredactions

qtwiredactions.win

vrmakeobject

Declared In

`Movies.h`

ShowMovieInformation

Displays a movie's information.

```
void ShowMovieInformation (
   Movie theMovie,
   ModalFilterUPP filterProc,
   long refCon
);
```

Parameters

theMovie

A movie identifier. Your application obtains this identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

filterProc

A Universal Procedure Pointer that accesses a `ModalFilterProc` callback.

refCon

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

Return Value

You can access error returns from this function through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qtinfo

qtinfo.win

Declared In
`Movies.h`

SpriteHitTest

Determines whether a location in a sprite's display coordinate system intersects the sprite.

```
OSErr SpriteHitTest (
   Sprite theSprite,
   long flags,
   Point loc,
   Boolean *wasHit
);
```

Parameters

theSprite

The sprite for this operation.

flags

Specifies flags (see below) that control the hit testing operation. See these constants:

`spriteHitTestBounds`

`spriteHitTestImage`

`spriteHitTestInvisibleSprites`

`spriteHitTestIsClick`

`spriteHitTestLocInDisplayCoordinates`

`spriteHitTestTreatAllSpritesAsHitTestable`

loc

A point in the sprite world's display space to test for the existence of a sprite. You should apply the sprite world's matrix to the point before passing it to this function.

wasHit

A pointer to a Boolean. On return, the value of the Boolean is TRUE if the sprite is at the specified location.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
This function is useful for hit testing a subset of the sprites in a sprite world and for detecting multiple hits for a single location.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

SpriteWorldHitTest

Determines whether any sprites are at a specified location in a sprite world.

```
OSErr SpriteWorldHitTest (
   SpriteWorld theSpriteWorld,
   long flags,
   Point loc,
   Sprite *spriteHit
);
```

Parameters

theSpriteWorld
The sprite world for this operation.

flags
Specifies flags (see below) that control the hit testing operation. See these constants:
`spriteHitTestBounds`
`spriteHitTestImage`
`spriteHitTestInvisibleSprites`
`spriteHitTestIsClick`
`spriteHitTestLocInDisplayCoordinates`
`spriteHitTestTreatAllSpritesAsHitTestable`

loc
A point in the sprite world's display space to test for the existence of a sprite.

spriteHit
A pointer to a field that is to receive a sprite identifier. On return, this field contains the identifier of the frontmost sprite at the location specified by the `loc` parameter. If no sprite exists at the location, the function sets the `value` of this parameter to `NIL`.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
If you are drawing the sprite world in a window, you should convert the location to your window's local coordinate system before passing it to `SpriteWorldHitTest`. A hit testing operation does not occur unless you pass either `spriteHitTestBounds` or `spriteHitTestImage` in the `flags` parameter. You can add other flags as needed.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

SpriteWorldIdle

Allows a sprite world to update its invalid areas.

```
OSErr SpriteWorldIdle (
   SpriteWorld theSpriteWorld,
   long flagsIn,
   long *flagsOut
);
```

Parameters

`theSpriteWorld`
: The sprite world for this operation.

`flagsIn`
: Contains flags (see below) describing actions that may take place during the idle. For the default behavior, set this parameter to 0. See these constants:
  `kOnlyDrawToSpriteWorld`

`flagsOut`
: On return, a pointer to flags (see below) describing actions that took place during the idle period. This parameter is optional; if you do not need the information, set it to `NIL`. See these constants:
  `kSpriteWorldDidDraw`
  `kSpriteWorldNeedsToDraw`

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

This is the only sprite function that causes drawing to occur; you should call it as often as is necessary. Typically, you would make changes in perspective for a number of sprites and then call `SpriteWorldIdle` to redraw the changed sprites.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Desktop Sprites

DesktopSprites

DesktopSprites.win

Declared In

`Movies.h`

UpdateMovieInStorage

Updates a movie at a storage location.

```
OSErr UpdateMovieInStorage (
   Movie theMovie,
   DataHandler dh
);
```

Parameters

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

dh

The data handler component that was returned by `CreateMovieStorage` (page 1318).

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

This function, which is similar to `OpenMovieStorage` (page 1417), replaces the content of the movie in the storage associated with the specified data handler.

Version Notes

Introduced in QuickTime 6. Supersedes `UpdateMovieResource` (page 1503).

Availability

Available in Mac OS X v10.2 and later.

Related Sample Code

QTCarbonShell

Declared In

`Movies.h`

UpdateMovieResource

Replaces the contents of a movie resource in a specified movie file.

```
OSErr UpdateMovieResource (
   Movie theMovie,
   short resRefNum,
   short resId,
   ConstStr255Param resName
);
```

Parameters

theMovie

The movie you wish to place in the movie file. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

resRefNum
Identifies the movie file that contains the resource to be changed. Your application obtains this value from `OpenMovieFile` (page 1416).

resId
The resource to be changed. This value is obtained from a previous call to `NewMovieFromFile` (page 1398), `NewMovieFromDataRef` (page 1397), or `AddMovieResource` (page 1299). If you specify a single-fork movie file by passing the `movieInDataForkResID` constant, the Movie Toolbox places the movie resource into the file's data fork.

resName
Points to a new name for the resource. If you don't want to change the resource's name, set this parameter to `NIL`.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
You specify the movie that is to be placed into the resource. This function can accommodate single-fork movie files. After updating the movie file, this function clears the movie changed flag.

Version Notes
Introduced in QuickTime 3 or earlier. Superseded in QuickTime 6 by `UpdateMovieInStorage` (page 1503).

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
BurntTextSampleCode
ChromaKeyMovie
MakeEffectMovie
qtwiredactions
qtwiredactions.win

Declared In
`Movies.h`

Callbacks

GetMovieProc

Provides movie data to the Movie Toolbox.

```
typedef OSErr (*GetMovieProcPtr) (long offset, long size, void *dataPtr, void
*refCon);
```

If you name your function `MyGetMovieProc`, you would declare it this way:

```
OSErr MyGetMovieProc (
    long    offset,
    long    size,
    void    *dataPtr,
    void     *refCon );
```

Parameters

offset
: Specifies the offset into the movie resource (not the movie file). This is the location from which your function retrieves the movie data.

size
: Specifies the amount of data requested by the toolbox, in bytes.

dataPtr
: Specifies the destination for the movie data.

refCon
: Contains a reference constant (defined as a void pointer). This is the same value you provided to the toolbox when you called `NewMovieFromUserProc` (page 1404).

Return Value
See `Error Codes`. Your callback should return `noErr` if there is no error.

Discussion
Normally, when a movie is loaded from a file (for example, by means of the `NewMovieFromFile` function), the toolbox uses that file as the default data reference. Since `NewMovieFromUserProc` (page 1404) does not require a file specification, your application should specify the file to be used as the default data reference using the `defaultDataRef` and `dataRefType` parameters.

Special Considerations

The toolbox automatically sets the movie's graphics world based upon the current graphics port. Be sure that your application's graphics world is valid before you call this function.

Declared In
`Movies.h`

MovieExecuteWiredActionsProc

Undocumented

```
typedef OSErr (*MovieExecuteWiredActionsProcPtr) (Movie theMovie, void *refcon,
long flags, QTAtomContainer wiredActions);
```

If you name your function `MyMovieExecuteWiredActionsProc`, you would declare it this way:

```
OSErr MyMovieExecuteWiredActionsProc (
    Movie              theMovie,
    void               *refcon,
    long               flags,
    QTAtomContainer    wiredActions );
```

Parameters

theMovie
: Specifies the movie for this operation.

refcon
: Pointer to a reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

flags

Undocumented

wiredActions

Undocumented

Return Value
See `Error Codes`. Your callback should return `noErr` if there is no error.

Declared In
`Movies.h`

MovieRgnCoverProc

Undocumented

```
typedef OSErr (*MovieRgnCoverProcPtr) (Movie theMovie, RgnHandle changedRgn, long
 refcon);
```

If you name your function `MyMovieRgnCoverProc`, you would declare it this way:

```
OSErr MyMovieRgnCoverProc (
    Movie        theMovie,
    RgnHandle    changedRgn,
    long         refcon );
```

Parameters

theMovie

Specifies the movie for this operation.

changedRgn

Undocumented

refcon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

Return Value
See `Error Codes`. Your callback should return `noErr` if there is no error.

Declared In
`Movies.h`

QTEffectListFilterProc

Called for each effect which passes the other criteria for inclusion in the effects list, and returns TRUE if the effect is to be included in the list.

```
typedef Boolean (*QTEffectListFilterProcPtr) (Component effect,
long effectMinSource, long effectMaxSource, OSType majorClass,
OSType minorClass, void *refcon);
```

If you name your function `MyQTEffectListFilterProc`, you would declare it this way:

```
Boolean MyQTEffectListFilterProc (
    Component    effect,
    long         effectMinSource,
    long         effectMaxSource,
    OSType       majorClass,
    OSType       minorClass,
    void         *refcon );
```

Parameters

effect
: The effect component.

effectMinSource
: The minimum number of sources that an effect must have to be added to the list. Pass -1 to specify no minimum.

effectMaxSource
: The maximum number of sources that an effect can have to be added to the list. Pass -1 to specify no maximum.

majorClass
: The major class to include, or 0 for all.

minorClass
: The minor class to include, or 0 for all.

refcon
: A reference constant that points to a data structure containing information the callback needs.

Return Value

See `Error Codes`. Your callback should return `noErr` if there is no error.

Discussion

Note that your filter `proc` may receive multiple effects from various manufacturers. If you return TRUE for multiple effects of a given type, only the one with the higher parameter version number will be included. If you wish other filtering such as effects from a given manufacturer, you can do this by returning FALSE for the other effects and TRUE for those that you prefer.

Declared In

`Movies.h`

QTSyncTaskProc

Undocumented

```
typedef void (*QTSyncTaskProcPtr) (void *task);
```

If you name your function `MyQTSyncTaskProc`, you would declare it this way:

```
void MyQTSyncTaskProc (
    void    *task );
```

Parameters

task
: *Undocumented*

Declared In

`Movies.h`

TweenerDataProc

A callback the tween component calls with the value generated by a tween operation.

```
typedef ComponentResult (*TweenerDataProcPtr) (TweenRecord *tr, void *tweenData,
long tweenDataSize, long dataDescriptionSeed, Handle dataDescription,
ICMCompletionProcRecordPtr asyncCompletionProc, UniversalProcPtr transferProc, void
 *refCon);
```

If you name your function `MyTweenerDataProc`, you would declare it this way:

```
ComponentResult MyTweenerDataProc (
    TweenRecord                   *tr,
    void                          *tweenData,
    long                          tweenDataSize,
    long                          dataDescriptionSeed,
    Handle                        dataDescription,
    ICMCompletionProcRecordPtr    asyncCompletionProc,
    UniversalProcPtr              transferProc,
    void                          *refCon );
```

Parameters

tr

A pointer to the tween record for the tween operation.

tweenData

A pointer to the generated tween value.

tweenDataSize

The size, in bytes, of the tween value.

dataDescriptionSeed

The starting value for the calculation. Every time the content of the `dataDescription` handle changes, this value should be incremented.

dataDescription

Specifies a handle containing a description of the tween value passed. For basic types such as integers, the calling tween component should set this parameter to `NIL`. For more complex types such as compressed image data, the calling tween component should set this handle to contain a description of the tween value, such as an image description.

asyncCompletionProc

A pointer to a completion procedure for asynchronous operations. The calling tween component should set the `value` of this parameter to `NIL`.

transferProc

A pointer to a procedure to transfer the data. The calling tween component should set the `value` of this parameter to `NIL`.

refCon

A pointer to a reference constant. The calling tween component should set the `value` of this parameter to `NIL`.

Return Value

See `Error Codes`. Your callback should return `noErr` if there is no error.

Declared In

`Movies.h`

Data Types

FourCharCode

Represents a type used by the Movie Toolkit API.

```
typedef unsigned long FourCharCode;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`IOHIDDescriptorParser.h`

FSSpecPtr

Represents a type used by the Movie Toolkit API.

```
typedef FSSpec * FSSpecPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Files.h`

GetMovieUPP

Represents a type used by the Movie Toolkit API.

```
typedef STACK_UPP_TYPE(GetMovieProcPtr) GetMovieUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

MovieExecuteWiredActionsUPP

Represents a type used by the Movie Toolkit API.

```
typedef STACK_UPP_TYPE(MovieExecuteWiredActionsProcPtr) MovieExecuteWiredActionsUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

MovieRgnCoverUPP

Represents a type used by the Movie Toolkit API.

```
typedef STACK_UPP_TYPE(MovieRgnCoverProcPtr) MovieRgnCoverUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

QTAtomType

Represents a type used by the Movie Toolkit API.

```
typedef long QTAtomType;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.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`

Discussion
The number of audio channels.

`numFrequencyBands`

Discussion
The number of frequency bands for each channel.

`level`

Discussion
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`.

Related Functions
Associated function: `GetMovieAudioFrequencyLevels` (page 1351)

Declared In
`Movies.h`

QTAudioVolumeLevels

Stores the volume level settings for the audio channels in a movie mix.

```
struct QTAudioVolumeLevels {
 UInt32     numChannels;
 Float32    level[1];
};
```

Fields

`numChannels`

Discussion

The number of audio channels.

`level`

Discussion

A 32-bit floating-point value for each channel's volume.

Related Functions

Associated function: `GetMovieAudioVolumeLevels` (page 1354)

Declared In

`Movies.h`

QTEffectListFilterUPP

Represents a type used by the Movie Toolkit API.

```
typedef STACK_UPP_TYPE(QTEffectListFilterProcPtr) QTEffectListFilterUPP;
```

Availability

Available in Mac OS X v10.2 and later.

Declared In

`Movies.h`

QTEffectListOptions

Represents a type used by the Movie Toolkit API.

```
typedef long QTEffectListOptions;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

QTErrorReplacementPtr

Represents a type used by the Movie Toolkit API.

```
typedef QTErrorReplacementRecord * QTErrorReplacementPtr;
```

Availability
Available in Mac OS X v10.2 and later.

Declared In
`Movies.h`

QTErrorReplacementRecord

Contains the list of strings to subsitute for variables in an error message.

```
struct QTErrorReplacementRecord {
     long         numEntries;
     StringPtr    replacementString[1];
 };
```

Fields
`numEntries`
Discussion
The number of string pointers in `replacementString`.

`replacementString`
Discussion
An array of string pointers. Memory for each string is allocated separately.

Version Notes
Introduced in QuickTime 6.

Related Functions
`QTAddMovieError` (page 1420)

Declared In
`Movies.h`

QTRestrictionSet

Represents a type used by the Movie Toolkit API.

```
typedef QTRestrictionSetRecord * QTRestrictionSet;
```

Availability
Available in Mac OS X v10.2 and later.

Declared In
`Movies.h`

QTRestrictionSetRecord

Holds a movie's restrictions.

```
struct QTRestrictionSetRecord {
     long    data[1];
 };
```

Fields
`data`

Discussion
The restrictions for a movie. See `Movie Restrictions`.

Version Notes
Introduced in QuickTime 6.

Related Functions
`QTGetMovieRestrictions` (page 1444)
`QTRestrictionsGetIndClass` (page 1463)
`QTRestrictionsGetInfo` (page 1464)
`QTRestrictionsGetItem` (page 1465)

Declared In
`Movies.h`

QTSyncTaskUPP

Represents a type used by the Movie Toolkit API.

```
typedef STACK_UPP_TYPE(QTSyncTaskProcPtr) QTSyncTaskUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

QTTweener

Represents a type used by the Movie Toolkit API.

```
typedef QTTweenerRecord * QTTweener;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

QTTweenerRecord

Stores a tween for the QTNewTween function.

```
struct QTTweenerRecord {
     long    data[1];
 };
```

Fields

`data`

Discussion

An array of data that constitutes a tween.

Declared In

`Movies.h`

QTUUID

Contains QuickTime's version of a universally unique identifier.

```
struct QTUUID {
     UInt32    data1;
     UInt16    data2;
     UInt16    data3;
     UInt8     data4[8];
 };
```

Fields

`data1`

Discussion

Undocumented

`data2`

Discussion

Undocumented

`data3`

Discussion

Undocumented

`data4`

Discussion

Undocumented

Version Notes

Introduced in QuickTime 6.

Related Functions

`QTCreateUUID` (page 1426)
`QTEqualUUIDs` (page 1430)

Declared In

`Movies.h`

Sprite

Represents a type used by the Movie Toolkit API.

```
typedef SpriteRecord * Sprite;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

SpriteRecord

Contains a sprite.

```
struct SpriteRecord {
     long    data[1];
 };
```

Fields
`data`

Discussion
An array of sprite data.

Declared In
`Movies.h`

SpriteWorld

Represents a type used by the Movie Toolkit API.

```
typedef SpriteWorldRecord * SpriteWorld;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

SpriteWorldRecord

Contains a sprite world.

```
struct SpriteWorldRecord {
     long    data[1];
 };
```

Fields
`data`

Discussion
An array of sprite world data.

Declared In
`Movies.h`

TweenerDataUPP

Represents a type used by the Movie Toolkit API.

```
typedef STACK_UPP_TYPE(TweenerDataProcPtr) TweenerDataUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

Constants

SetQuickTimePreference Values

Constants passed to SetQuickTimePreference.

```
enum {
  BandwidthManagementPrefsType  = 'bwmg'
};
```

Declared In
`Movies.h`

CreateMovieFile Values

Constants passed to CreateMovieFile.

```
enum {
  createMovieFileDeleteCurFile  = 1L << 31,
  createMovieFileDontCreateMovie = 1L << 30,
  createMovieFileDontOpenFile   = 1L << 29,
  createMovieFileDontCreateResFile = 1L << 28
};
```

Constants
`createMovieFileDontOpenFile`

Controls whether the function opens the new movie file. If you set this flag to 1, the Movie Toolbox does not open the new movie file. In this case, the function ignores the outDataHandler parameter. If you set this flag to 0, the Movie Toolbox opens the new movie file and returns its reference number into the field referenced by outDataHandler.

Available in Mac OS X v10.0 and later.

Declared in `Movies.h`.

Declared In
`Movies.h`

GetMediaDataRef Values

Constants passed to GetMediaDataRef.

```
enum {
  dataRefSelfReference          = 1 << 0,
  dataRefWasNotResolved         = 1 << 1
};
```

Declared In
`Movies.h`

QTGetEffectSpeed Values

Constants passed to QTGetEffectSpeed.

```
enum {
  effectIsRealtime              = 0     /* effect can be rendered in real time */
};
```

Declared In
`Movies.h`

QTGetEffectsList Values

Constants passed to QTGetEffectsList.

```
enum {
  elOptionsIncludeNoneInList    = 0x00000001 /* "None" effect is included in list
 */
};
```

Declared In
`Movies.h`

Full Screen Flags

Constants that represent flags for full screen displays.

```
enum {
  fullScreenHideCursor          = 1L << 0,
  fullScreenAllowEvents         = 1L << 1,
  fullScreenDontChangeMenuBar   = 1L << 2,
  fullScreenPreflightSize       = 1L << 3,
  fullScreenDontSwitchMonitorResolution = 1L << 4,
  fullScreenCaptureDisplay      = 1 << 5L, /* capturedisplay is a mac os x specific
 parameter */
  fullScreenCaptureAllDisplays  = 1 << 6L /* capturealldisplays is a mac os x
specific parameter */
};
```

Constants

`fullScreenHideCursor`

If this flag is set, `BeginFullScreen` hides the cursor. This is useful if you are going to play a QuickTime movie and do not want the cursor to be visible over the movie.

Available in Mac OS X v10.0 and later.

Declared in `Movies.h`.

`fullScreenAllowEvents`

If this flag is set, your application intends to allow other applications to run (by calling `WaitNextEvent` to grant them processing time). In this case, `BeginFullScreen` does not change the monitor resolution, because other applications might depend on the current resolution.

Available in Mac OS X v10.0 and later.

Declared in `Movies.h`.

`fullScreenDontChangeMenuBar`

If this flag is set, `BeginFullScreen` does not hide the menu bar. This is useful if you want to change the resolution of the monitor but still need to allow the user to access the menu bar.

Available in Mac OS X v10.0 and later.

Declared in `Movies.h`.

`fullScreenPreflightSize`

If this flag is set, `BeginFullScreen` doesn't change any monitor settings, but returns the actual height and width that it would use if this bit were not set. This allows applications to test for the availability of a monitor setting without having to switch to it.

Available in Mac OS X v10.0 and later.

Declared in `Movies.h`.

`fullScreenCaptureDisplay`

`Capturedisplay` is a Mac OS X specific parameter.

Available in Mac OS X v10.3 and later.

Declared in `Movies.h`.

Declared In

`Movies.h`

Hint Flags

Constants that represent hint flags.

```
enum {
  hintsScrubMode                = 1 << 0, /* mask == && (if flags == scrub on,
flags != scrub off) */
  hintsLoop                     = 1 << 1,
  hintsDontPurge                = 1 << 2,
  hintsUseScreenBuffer          = 1 << 5,
  hintsAllowInterlace           = 1 << 6,
  hintsUseSoundInterp           = 1 << 7,
  hintsHighQuality              = 1 << 8, /* slooooow */
  hintsPalindrome               = 1 << 9,
  hintsInactive                 = 1 << 11,
  hintsOffscreen                = 1 << 12,
  hintsDontDraw                 = 1 << 13,
  hintsAllowBlacklining         = 1 << 14,
  hintsDontUseVideoOverlaySurface = 1 << 16,
  hintsIgnoreBandwidthRestrictions = 1 << 17,
  hintsPlayingEveryFrame        = 1 << 18,
  hintsAllowDynamicResize       = 1 << 19,
  hintsSingleField              = 1 << 20,
  hintsNoRenderingTimeOut       = 1 << 21,
  hintsFlushVideoInsteadOfDirtying = 1 << 22,
  hintsEnableSubPixelPositioning = 1L << 23,
  hintsRenderingMode            = 1L << 24,
  hintsAllowIdleSleep           = 1L << 25, /* asks media handlers not to call
UpdateSystemActivity etc */
  hintsDeinterlaceFields        = 1L << 26
};
```

Constants

`hintsAllowIdleSleep`

Asks media handlers not to call `UpdateSystemActivity` etc.

Available in Mac OS X v10.3 and later.

Declared in `Movies.h`.

Declared In

`Movies.h`

QTUnregisterAccessKey Values

Constants passed to QTUnregisterAccessKey.

```
enum {
  kAccessKeySystemFlag          = 1L << 0
};
```

Declared In

`Movies.h`

Sprite Properties

Constants that represent the properties of sprites.

```
enum {
  kGetSpriteWorldInvalidRegionAndLeaveIntact = -1L,
  kGetSpriteWorldInvalidRegionAndThenSetEmpty = -2L
};
enum {
  kKeyFrameAndSingleOverride    = 1L << 1,
  kKeyFrameAndAllOverrides      = 1L << 2
};
enum {
  kNoQTIdleEvents               = -1
};
enum {
  kOnlyDrawToSpriteWorld        = 1L << 0,
  kSpriteWorldPreflight         = 1L << 1
};
enum {
  kScaleSpritesToScaleWorld     = 1L << 1,
  kSpriteWorldHighQuality       = 1L << 2,
  kSpriteWorldDontAutoInvalidate = 1L << 3,
  kSpriteWorldInvisible         = 1L << 4,
  kSpriteWorldDirtyInsteadOfFlush = 1L << 5
};
enum {
  kSpritePropertyMatrix         = 1,
  kSpritePropertyImageDescription = 2,
  kSpritePropertyImageDataPtr   = 3,
  kSpritePropertyVisible        = 4,
  kSpritePropertyLayer          = 5,
  kSpritePropertyGraphicsMode   = 6,
  kSpritePropertyImageDataSize  = 7,
  kSpritePropertyActionHandlingSpriteID = 8,
  kSpritePropertyCanBeHitTested = 9,
  kSpritePropertyImageIndex     = 100,
  kSpriteTrackPropertyBackgroundColor = 101,
  kSpriteTrackPropertyOffscreenBitDepth = 102,
  kSpriteTrackPropertySampleFormat = 103,
  kSpriteTrackPropertyScaleSpritesToScaleWorld = 104,
  kSpriteTrackPropertyHasActions = 105,
  kSpriteTrackPropertyVisible   = 106,
  kSpriteTrackPropertyQTIdleEventsFrequency = 107,
  kSpriteTrackPropertyAllSpritesHitTestingMode = 108,
  kSpriteTrackPropertyPreferredDepthInterpretationMode = 109,
  kSpriteImagePropertyRegistrationPoint = 1000,
  kSpriteImagePropertyGroupID   = 1001
};
```

Declared In
`Movies.h`

SetMediaDataRefAttributes Values

Constants passed to SetMediaDataRefAttributes.

```
enum {
  kMovieAnchorDataRefIsDefault  = 1 << 0 /* data ref returned is movie default data
 ref */
};
```

Declared In
`Movies.h`

CopyUserData Values

Constants passed to CopyUserData.

```
enum {
  kQTCopyUserDataReplace        = 'rplc', /* Delete all destination user data items
 and then add source user data items */
  kQTCopyUserDataMerge         = 'merg' /* Add source user data items to destination
 user data */
};
```

Declared In
`Movies.h`

CanQuickTimeOpenFile Values

Constants passed to CanQuickTimeOpenFile.

```
enum {
  kQTDontUseDataToFindImporter  = 1L << 0,
  kQTDontLookForMovieImporterIfGraphicsImporterFound = 1L << 1,
  kQTAllowOpeningStillImagesAsMovies = 1L << 2,
  kQTAllowImportersThatWouldCreateNewFile = 1L << 3,
  kQTAllowAggressiveImporters   = 1L << 4 /* eg, TEXT and PICT movie importers*/
};
```

Declared In
`Movies.h`

QTNewDataReferenceFromFullPathCFString Values

Constants passed to QTNewDataReferenceFromFullPathCFString.

```
enum {
  kQTNativeDefaultPathStyle     = -1,
  kQTPOSIXPathStyle             = 0,
  kQTHFSPathStyle               = 1,
  kQTWindowsPathStyle           = 2
};
```

Declared In
`Movies.h`

SpriteWorldIdle Values

Constants passed to SpriteWorldIdle.

```
enum {
  kSpriteWorldDidDraw           = 1L << 0,
  kSpriteWorldNeedsToDraw       = 1L << 1
};
```

Declared In
`Movies.h`

MovieExecuteWiredActions Values

Constants passed to MovieExecuteWiredActions.

```
enum {
  movieExecuteWiredActionDontExecute = 1L << 0
};
```

Declared In
`Movies.h`

NewMovieFromFile Values

Constants passed to NewMovieFromFile.

```
enum {
  movieInDataForkResID          = -1    /* magic res ID */
};
```

Declared In
`Movies.h`

PutMovieOnScrap Values

Constants passed to PutMovieOnScrap.

```
enum {
  movieScrapDontZeroScrap       = 1 << 0,
  movieScrapOnlyPutMovie        = 1 << 1
};
```

Declared In
`Movies.h`

SetTrackLoadSettings Values

Constants passed to SetTrackLoadSettings.

```
enum {
  preloadAlways                   = 1L << 0,
  preloadOnlyIfEnabled            = 1L << 1
};
```

Declared In
`Movies.h`

MovieSearchText Values

Constants passed to MovieSearchText.

```
enum {
  searchTextDontGoToFoundTime   = 1L << 16,
  searchTextDontHiliteFoundText = 1L << 17,
  searchTextOneTrackOnly        = 1L << 18,
  searchTextEnabledTracksOnly   = 1L << 19
};
```

Declared In
`Movies.h`

Media Characteristics

Constants that represent the characteristics of media.

```
enum {
  VisualMediaCharacteristic     = 'eyes',
  AudioMediaCharacteristic      = 'ears',
  kCharacteristicCanSendVideo   = 'vsnd',
  kCharacteristicProvidesActions = 'actn',
  kCharacteristicNonLinear      = 'nonl',
  kCharacteristicCanStep        = 'step',
  kCharacteristicHasNoDuration  = 'noti',
  kCharacteristicHasSkinData    = 'skin',
  kCharacteristicProvidesKeyFocus = 'keyf',
  kCharacteristicSupportsDisplayOffsets = 'dtdd'
};
```

Constants

`AudioMediaCharacteristic`

Value = `'ears'`. Instructs the Movie Toolbox to search all tracks that play sound.

Available in Mac OS X v10.0 and later.

Declared in `Movies.h`.

Declared In
`Movies.h`

QuickTime Movie Track and Media Reference

Framework:	Frameworks/QuickTime.framework
Declared in	Movies.h

Overview

Track and media management functions help with the construction and editing of QuickTime movies.

Functions by Task

Adding Samples to Media Structures

`AddMediaSample` (page 1536)
    Adds sample data and a description to a media.

`AddMediaSampleReference` (page 1541)
    Works with samples that have already been added to a movie data file.

`AddMediaSampleReferences` (page 1543)
    Adds groups of samples to a movie data file.

`BeginMediaEdits` (page 1549)
    Starts a media-editing session.

`EndMediaEdits` (page 1567)
    Ends a media-editing session.

`GetMediaPreferredChunkSize` (page 1582)
    Retrieves the maximum chunk size for a media.

`GetMediaSample` (page 1583)
    Returns a sample from a movie data file.

`GetMediaSampleReference` (page 1589)
    Obtains reference information about samples that are stored in a movie data file.

`GetMediaSampleReferences` (page 1590)
    Obtains reference information about groups of samples that are stored in a movie.

`SetMediaDefaultDataRefIndex` (page 1643)
    Specifies which of a media's data references is to be accessed during an editing session.

`SetMediaPreferredChunkSize` (page 1647)
    Specifies a maximum chunk size for a media.

Creating Tracks and Media Structures

`DisposeMovieTrack` (page 1564)
Removes a track from a movie.

`DisposeTrackMedia` (page 1566)
Removes a media from a track.

`NewMovieTrack` (page 1628)
Creates a new movie track, without a media.

`NewTrackMedia` (page 1630)
Creates a media for a new track.

Determining Movie Creation and Modification Time

`GetMediaCreationTime` (page 1570)
Returns the creation date and time stored in a media.

`GetMediaModificationTime` (page 1581)
Returns a media's modification date and time.

`GetTrackCreationTime` (page 1604)
Returns a track's creation date and time.

`GetTrackModificationTime` (page 1613)
Returns a track's modification date and time.

Disabling Movies and Tracks

`GetTrackEnabled` (page 1609)
Determines whether a track is currently enabled.

`SetTrackEnabled` (page 1651)
Enables or disables a track.

Editing Tracks

`AddEmptyTrackToMovie` (page 1535)
Duplicates a track from a movie into the same movie or into another movie.

`CopyTrackSettings` (page 1559)
Copies many settings from one track to another, overwriting the destination settings.

`DeleteTrackSegment` (page 1563)
Removes a specified segment from a track.

`GetTrackEditRate` (page 1608)
Returns the rate of the track edit of a specified track at an indicated time.

`InsertEmptyTrackSegment` (page 1619)
Adds an empty segment to a track.

`InsertMediaIntoTrack` (page 1620)
Inserts a reference to a media segment into a track.

InsertTrackSegment (page 1623)
Copies data into a track.

ScaleTrackSegment (page 1641)
Changes the duration of a segment of a track.

Enhancing Movie Playback Performance

GetMediaShadowSync (page 1593)
Obsolete; no longer supported.

GetTrackDisplayMatrix (page 1607)
Returns a matrix that is the concatenation of all matrices currently affecting the track's location, scaling, and so on, including the movie's matrix, the track's matrix, and the modifier matrix.

SetMediaShadowSync (page 1649)
Obsolete; no longer supported.

Finding and Adding Samples

AddMediaSample2 (page 1539)
Adds sample data and a description to a media.

ExtendMediaDecodeDurationToDisplayEndTime (page 1568)
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.

GetMediaAdvanceDecodeTime (page 1570)
Returns the advance decode time of a media.

GetMediaDataSizeTime64 (page 1573)
Determines the size, in bytes, of the sample data in a media segment.

GetMediaDecodeDuration (page 1574)
Returns the decode duration of a media.

GetMediaDisplayDuration (page 1575)
Returns the display duration of a media.

GetMediaDisplayEndTime (page 1575)
Returns the display end time of a media.

GetMediaDisplayStartTime (page 1576)
Returns the display start time of a media.

MediaContainsDisplayOffsets (page 1625)
Tests whether a media contains display offsets.

MediaDecodeTimeToSampleNum (page 1625)
Finds the sample for a specified decode time.

MediaDisplayTimeToSampleNum (page 1626)
Finds the sample number for a specified display time.

TrackTimeToMediaDisplayTime (page 1658)
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.

High-Level Movie Editing Functions

`AddMovieSelection` (page 1545)
Adds one or more tracks to a movie.

`ClearMovieSelection` (page 1550)
Removes the segment of the movie that is defined by the current selection.

`CopyMovieSelection` (page 1558)
Creates a new movie that contains the original movie's current selection.

`CutMovieSelection` (page 1560)
Creates a new movie that contains the original movie's current selection.

`IsScrapMovie` (page 1624)
Checks the system scrap to find out if it can translate any of the data into a movie.

`PasteHandleIntoMovie` (page 1633)
Takes the contents of a specified handle, together with its type, and pastes it into a specified movie.

`PasteMovieSelection` (page 1634)
Places the tracks from one movie into another movie.

`PutMovieIntoTypedHandle` (page 1636)
Takes a movie, or a single track from within that movie, and converts it into a handle of a specified type.

Locating a Movie's Tracks and Media Structures

`GetMediaTrack` (page 1595)
Determines the track that uses a specified media.

`GetMovieIndTrack` (page 1598)
Determines the track identifier of a track, given the track's index value.

`GetMovieIndTrackType` (page 1600)
Searches for all of a movie's tracks that share a given media type or media characteristic.

`GetMovieTrack` (page 1601)
Determines the track identifier of a track, given the track's ID value.

`GetMovieTrackCount` (page 1601)
Returns the number of tracks in a movie.

`GetTrackID` (page 1610)
Determines a track's unique track ID value.

`GetTrackMedia` (page 1612)
Determines the media that contains a track's sample data.

`GetTrackMovie` (page 1613)
Determines the movie that contains a specified track.

Low-Level Movie Editing Functions

`CopyMovieSettings` (page 1558)
Copies many settings from one movie to another, overwriting the destination settings in the process.

`DeleteMovieSegment` (page 1561)
Removes a specified segment from a movie.

`InsertEmptyMovieSegment` (page 1618)
Adds an empty segment to a movie.

`InsertMovieSegment` (page 1622)
Copies part of one movie to another.

`ScaleMovieSegment` (page 1640)
Changes the duration of a segment of a movie.

Manipulating Media Input Maps

`GetMediaInputMap` (page 1579)
Returns a copy of the input map associated with a specified media.

`SetMediaInputMap` (page 1645)
Replaces the media's existing input map with a given input map.

Movie Functions

`ConvertFileToMovieFile` (page 1552)
Converts a file to a movie file and supports a user settings dialog box for import operations.

`ConvertMovieToFile` (page 1555)
Takes a specified movie (or a single track within that movie) and converts it into a specified file and type, supporting a Save As dialog box.

Movie Posters and Movie Previews

`GetTrackUsage` (page 1616)
Determines whether a track is used in a movie, its preview, its poster, or a combination of these.

`SetTrackUsage` (page 1656)
Specifies whether a track is used in a movie, its preview, its poster, or a combination of these.

Movies and Your Event Loop

`PtInMovie` (page 1634)
Determines whether a specified point lies in the region defined by a movie's final display boundary region after it has been clipped by the movie's display clipping region.

`PtInTrack` (page 1635)
Determines whether a specified point lies in the region defined by a track's display boundary region after it has been clipped by the movie's final display clipping region.

Selecting Media Handlers

`GetDataHandler` (page 1569)
Retrieves the best data handler component to use with a given data reference.

`GetMediaDataHandler` (page 1571)
Determines a media's data handler.

`GetMediaDataHandlerDescription` (page 1571)
Retrieves information about a media's data handler.

`GetMediaHandler` (page 1577)
Obtains a reference to a media handler component.

`GetMediaHandlerDescription` (page 1578)
Retrieves information about a media handler.

`SetMediaDataHandler` (page 1643)
Assigns a data handler to a media.

`SetMediaHandler` (page 1644)
Assigns a specific media handler to a track.

Undo for Movies

`DisposeMovieEditState` (page 1564)
Disposes of an edit state.

`NewMovieEditState` (page 1628)
Creates an edit state.

`UseMovieEditState` (page 1659)
Returns a movie to the condition determined by an edit state created previously.

Undo for Tracks

`DisposeTrackEditState` (page 1566)
Disposes of a movie's track edit state.

`NewTrackEditState` (page 1630)
Creates a new edit state for a given track.

`UseTrackEditState` (page 1660)
Returns a track to the condition determined by an edit state created previously.

Working With Alternate Tracks

`GetMediaLanguage` (page 1581)
Returns a media's localized language or region code.

`GetMediaQuality` (page 1582)
Returns a media's quality level value.

`GetTrackAlternate` (page 1603)
Determines all the tracks in an alternate group.

SelectMovieAlternates (page 1642)
Instructs the Movie Toolbox to select appropriate tracks immediately.

SetAutoTrackAlternatesEnabled (page 1642)
Enables or disables automatic track selection by the Movie Toolbox.

SetMediaLanguage (page 1646)
Sets a media's localized language or region code.

SetMediaQuality (page 1647)
Sets a media's quality level value.

SetTrackAlternate (page 1650)
Adds tracks to, or remove tracks from, alternate groups.

Working With Media Samples

GetMediaDataSize (page 1572)
Determines the size, in bytes, of the sample data in a media segment.

GetMediaSampleCount (page 1586)
Determines the number of samples in a media.

GetMediaSampleDescription (page 1587)
Retrieves a SampleDescription structure from a media.

GetMediaSampleDescriptionCount (page 1588)
Returns the number of sample descriptions in a media.

GetMovieDataSize (page 1596)
Determines the size of the sample data in a segment of a movie.

GetTrackDataSize (page 1604)
Determines the size, in bytes, of the sample data in a segment of a track.

MediaTimeToSampleNum (page 1627)
Lets you find the sample that contains the data for a specified time.

SampleNumToMediaTime (page 1639)
Finds the time at which a specified sample plays.

SetMediaSampleDescription (page 1648)
Changes the contents of a particular SampleDescription structure of a specified media.

Working With Media Time

GetMediaDuration (page 1576)
Returns the duration of a media.

GetMediaTimeScale (page 1594)
Determines a media's time scale.

SetMediaTimeScale (page 1649)
Sets a media's time scale.

Working With Movie Spatial Characteristics

`GetTrackDimensions` (page 1606)
Determines a track's source rectangle.

`GetTrackLayer` (page 1611)
Retrieves a track's layer.

`GetTrackMatrix` (page 1611)
Retrieves a track's transformation matrix.

`SetTrackDimensions` (page 1650)
Establishes a track's source rectangle.

`SetTrackLayer` (page 1652)
Sets a track's layer.

`SetTrackMatrix` (page 1653)
Establishes a track's transformation matrix.

Working With QuickTime Sample Tables

`AddSampleTableToMedia` (page 1547)
Adds a sample table to a media.

`CopyMediaMutableSampleTable` (page 1557)
Obtains information about sample references in a media in the form of a sample table.

Working With Sound Volume

`GetTrackVolume` (page 1618)
Returns a track's current volume setting.

`SetTrackVolume` (page 1657)
Sets a track's current volume.

Working With Track References

`AddTrackReference` (page 1547)
Adds a new track reference to a track.

`DeleteTrackReference` (page 1562)
Removes a track reference from a track.

`GetNextTrackReferenceType` (page 1602)
Determines all of the track reference types that are defined for a given track.

`GetTrackReference` (page 1614)
Retrieves the track identifier contained in an existing track reference.

`GetTrackReferenceCount` (page 1615)
Determines how many track references of a given type exist for a track.

`SetTrackReference` (page 1654)
Modifies an existing track reference.

Working With Track Sound

`GetTrackSoundLocalizationSettings` (page 1616)
Returns a handle to a copy of the current 3D sound settings for a specified track.

`SetTrackSoundLocalizationSettings` (page 1655)
Applies 3D sound effect data to a track.

Working With Track Time

`GetTrackDuration` (page 1607)
Returns the duration of a track.

`GetTrackOffset` (page 1614)
Determines the time difference between the start of a track and the start of the movie that contains the track.

`SetTrackOffset` (page 1653)
Modifies the duration of the empty space that lies at the beginning of a track, thus changing the duration of the entire track.

`TrackTimeToMediaTime` (page 1658)
Converts a track's time value to a time value that is appropriate to the track's media, using the track's edit list.

Working With User Data

`GetMediaUserData` (page 1595)
Obtains access to a media's user data list.

`GetTrackUserData` (page 1617)
Obtains access to a track's user data list.

Supporting Functions

`AddClonedTrackToMovie` (page 1534)
Constructs a clone of an existing track in a movie.

`AddMediaSampleFromEncodedFrame` (page 1541)
Adds sample data and description from an encoded frame to a media.

`AddMediaSampleReferences64` (page 1544)
Provides a 64-bit version of AddMediaSampleReferences.

`ConvertDataRefToMovieDataRef` (page 1551)
Converts a piece of data in a storage location to a movie file format and stores it in another storage location, supporting a user settings dialog box for import operations.

`ConvertMovieToDataRef` (page 1553)
Converts a specified movie (or a single track within a movie) into a specified file format and stores it in a specified storage location.

`GetMediaDataSize64` (page 1573)
Provides a 64-bit version of GetMediaDataSize.

`GetMediaSample2` (page 1585)
: Retrieves sample data from a media file.

`GetMediaSampleReferences64` (page 1592)
: Provides a 64-bit version of GetMediaSampleReferences.

`GetMediaSyncSampleCount` (page 1594)
: Gets the number of sync samples in a media.

`GetMovieDataSize64` (page 1596)
: Provides a 64-bit version of GetMovieDataSize.

`GetMovieImporterForDataRef` (page 1597)
: Gets the movie importer component for a movie.

`GetTrackDataSize64` (page 1605)
: Provides a 64-bit version of GetTrackDataSize.

`GetTrackEditRate64` (page 1609)
: Returns the rate of the track edit of a specified track at an indicated time.

`OpenADataHandler` (page 1631)
: Opens a data handler component.

`QTGetMIMETypeInfo` (page 1637)
: Retrieves information about a particular MIME type.

`SampleNumToMediaDecodeTime` (page 1638)
: Finds the decode time for a specified sample.

`SampleNumToMediaDisplayTime` (page 1638)
: Finds the display time for a specified sample.

Functions

AddClonedTrackToMovie

Constructs a clone of an existing track in a movie.

```
OSErr AddClonedTrackToMovie (
   Track srcTrack,
   Movie dstMovie,
   long flags,
   Track *dstTrack
);
```

Parameters

sourceTrack
: Indicates the track to be cloned. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601). This is the source of the sample table once the cloned track is constructed.

destinationMovie
: Indicates the movie where the cloned track should be created. Your application obtains this identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400). Currently, this must be the movie that contains the source track.

flags

Flags (see below) that determine how cloning should be performed. You currently must pass `kQTCloneShareSamples`. See these constants:

`kQTCloneShareSamples`

`kQTCloneDontCopyEdits`

dstTrack

The address of storage where a reference to the newly constructed track is returned. If the function fails, this storage is set to `NIL`.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Special Considerations

Most QuickTime developers should never need to call this function.

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

AddEmptyTrackToMovie

Duplicates a track from a movie into the same movie or into another movie.

```
OSErr AddEmptyTrackToMovie (
   Track srcTrack,
   Movie dstMovie,
   Handle dataRef,
   OSType dataRefType,
   Track *dstTrack
);
```

Parameters

srcTrack

The source track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

dstMovie

The destination movie for this operation. This can be the same movie as the source track or a different movie.

dataRef

A handle to the `data` reference. The type of information stored in the handle depends upon the `data` reference type specified by `dataRefType`.

dataRefType

The type of data reference; see `Data References`. If the data reference is an alias, you must set the parameter to `rAliasType`, indicating that the reference is an alias.

dstTrack

The newly created track's identifier is returned in this parameter. If `AddEmptyTrackToMovie` fails, the resulting track identifier is set to `NIL`.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

This function returns a newly created, empty track. The newly created track has the same media type and track settings as the specified track. However, no data is copied from the source track to the new track. To copy data from the source track to the new track, use `InsertTrackSegment` (page 1623) after calling `AddEmptyTrackToMovie`.

Version Notes

This function has been available since QuickTime 2.0.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

bMoviePalette

bMoviePaletteCocoa

qtdataref

ThreadsImporter

ThreadsImportMovie

Declared In

`Movies.h`

AddMediaSample

Adds sample data and a description to a media.

```
OSErr AddMediaSample (
   Media theMedia,
   Handle dataIn,
   long inOffset,
   unsigned long size,
   TimeValue durationPerSample,
   SampleDescriptionHandle sampleDescriptionH,
   long numberOfSamples,
   short sampleFlags,
   TimeValue *sampleTime
);
```

Parameters

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (page 1630) and `GetTrackMedia` (page 1612). See `Media Identifiers`.

dataIn

A handle to the sample data. The `AddMediaSample` function adds this data to the media specified by the parameter `theMedia`. You specify the number of bytes of sample data with the `size` parameter. You can use the `inOffset` parameter to specify a byte offset into the data referred to by this handle.

inOffset
: Specifies an offset into the data referred to by the handle contained in the `dataIn` parameter. Set this parameter to 0 if there is no offset.

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.

durationPerSample
: The duration of each sample to be added. 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 to add a single sample.

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 (I/O). 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
: Contains flags (see below) that control the add operation. Set unused flags to 0. See these constants:
    `mediaSampleNotSync`

sampleTime
: A pointer to a time value. After adding the sample data to the media, the `AddMediaSample` function returns the time where the 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 `NIL`.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
Your application specifies the sample and the media for the operation. `AddMediaSample` updates the media so that it contains the sample data. One call to this function can add several samples to a media; however, all the samples must be the same size. Samples are always appended to the end of the media. Furthermore, the media duration is extended each time a sample is added.

```
// AddMediaSample coding example
// See "Discovering QuickTime," page 250
#define kSoundSampleDuration    1
#define kSyncSample             0
#define kTrackStart             0
#define kMediaStart             0
#define kFix1                   0x00010000
void CreateMySoundTrack (Movie movie)
{
    Track                       track;
    Media                       media;
    Handle                      hSound =NIL;
    SoundDescriptionHandle      hSoundDesc =NIL;
    long                        lDataOffset;
    long                        lDataSize;
    long                        lNumSamples;
    hSound =GetResource(soundListRsrc, 128);
    if (hSound ==NIL)
        return;
    hSoundDesc =(SoundDescriptionHandle)NewHandle(4);

    CreateMySoundDescription(hSound,
                            hSoundDesc,
                            &lDataOffset,
                            &lNumSamples,
                            &lDataSize);

    track =NewMovieTrack(movie, 0, 0, kFullVolume);
    media =NewTrackMedia(track, SoundMediaType,
                                FixRound((**hSoundDesc).sampleRate),
                                NIL, 0);
    BeginMediaEdits(media);
    AddMediaSample(media,
                    hSound,
                    lDataOffset,        // offset in data
                    lDataSize,
                    kSoundSampleDuration,   // duration of each sound
                                            //  sample
                    (SampleDescriptionHandle)hSoundDesc,
                    lNumSamples,
                    kSyncSample,        // self-contained samples
                    NIL);
    EndMediaEdits(media);
    InsertMediaIntoTrack(track,
                            kTrackStart,    // track start time
                            kMediaStart,    // media start time
                            GetMediaDuration(media),
                            kFix1);
    if (hSoundDesc !=NIL)
        DisposeHandle((Handle)hSoundDesc);
}
```

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qteffects

qteffects.win

vrmakepano

VRMakePano Library

vrmakepano.win

Declared In

`Movies.h`

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` (page 1630) and `GetTrackMedia` (page 1612).

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. See these constants:

`mediaSampleNotSync`

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` (page 221) and `GetMoviesStickyError` (page 222), 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` (page 1536); it adds 64-bit support and support for frame reordering video compression (display offset).

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

CaptureAndCompressIPBMovie

QTExtractAndConvertToMovieFile

QTKitTimeCode

SCAudioCompress

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` (page 1630) and `GetTrackMedia` (page 1612)

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` (page 221) and `GetMoviesStickyError` (page 222), 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:

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

OpenGLCaptureToMovie

Quartz Composer QCTV

Declared In

`Movies.h`

AddMediaSampleReference

Works with samples that have already been added to a movie data file.

```
OSErr AddMediaSampleReference (
   Media theMedia,
   long dataOffset,
   unsigned long size,
   TimeValue durationPerSample,
   SampleDescriptionHandle sampleDescriptionH,
   long numberOfSamples,
   short sampleFlags,
   TimeValue *sampleTime
);
```

Parameters

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (page 1630) and `GetTrackMedia` (page 1612). See `Media Identifiers`.

dataOffset

The offset into the movie data file. This parameter is used differently by each data handler. For example, for the standard HFS data handler, this parameter specifies the offset into the file. This parameter contains either data you add yourself or the data offset returned by `GetMediaSampleReference` (page 1589).

size

The number of bytes of sample data to be identified by the reference. This parameter indicates the total number of bytes in the sample data, not the number of bytes per sample. Use `numberOfSamples` to indicate the number of samples that are contained in the reference.

durationPerSample

The duration of each sample in the reference. You must specify this parameter in the media's time scale. For example, if you are referring to sound that was sampled at 22 kHz in a media that contains a sound track with the same time scale, to add a reference to a single sample you would set `durationPerSample` to 1. Similarly, if you are referring to video that was recorded at 10 frames per second in a video media that has a time scale of 60, you would set this parameter to 6 to add a reference to a single sample.

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 sound description structure. If the media does not require a `SampleDescription` structure, set this parameter to `NIL`.

numberOfSamples

The number of samples contained in the reference. For details, see `AddMediaSample` (page 1536). If the media does not require a `SampleDescription` structure, set this parameter to `NIL`.

sampleFlags

Contains flags (see below) that control the operation. Set unused flags to 0. See these constants:

`mediaSampleNotSync`

sampleTime

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

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

This function does not add sample data to the file or device that contains a media. Rather, it defines references to sample data that you previously added to a movie data file. Instead of actually writing out samples to disk, this function writes out references to existing samples, which you specify in `dataOffset` and the `size` parameter. As with `AddMediaSample` (page 1536), your application specifies the media for the operation. Note that one reference may refer to more than one sample; all the samples described by a reference must be the same size. This function does not update the movie data file as part of the add operation. Therefore, your application does not have to call `BeginMediaEdits` (page 1549) before calling `AddMediaSampleReference`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

AlwaysPreview

ElectricImageComponent.win

qt3dtween.win

SlideShowImporter

SlideShowImporter.win

Declared In

`Movies.h`

AddMediaSampleReferences

Adds groups of samples to a movie data file.

```
OSErr AddMediaSampleReferences (
   Media theMedia,
   SampleDescriptionHandle sampleDescriptionH,
   long numberOfSamples,
   SampleReferencePtr sampleRefs,
   TimeValue *sampleTime
);
```

Parameters

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (page 1630) and `GetTrackMedia` (page 1612). See `Media Identifiers`.

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 sound description structure. If you don't want the `SampleDescription` structure, set this parameter to `NIL`.

numberOfSamples

The number of `SampleReferenceRecord` structures pointed to by the `sampleRefs` parameter. Each structure may contain one or more contiguous samples. For details, see `AddMediaSample` (page 1536).

sampleRefs
A pointer to the number of `SampleReferenceRecord` structures specified by the `numberOfSamples` parameter.

sampleTime
A pointer to a time value. After adding the reference to the media, the `AddMediaSampleReferences` function returns the time where the reference was inserted, using the time scale referred to by this parameter. If you don't want to receive this information, set this parameter to `NIL`.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
Using this function instead of `AddMediaSampleReference` (page 1541) can greatly improve the performance of operations that involve adding a large number of samples to a movie at one time. `AddMediaSampleReferences` provides no capabilities that weren't previously available with `AddMediaSampleReference`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

AddMediaSampleReferences64

Provides a 64-bit version of AddMediaSampleReferences.

```
OSErr AddMediaSampleReferences64 (
   Media theMedia,
   SampleDescriptionHandle sampleDescriptionH,
   long numberOfSamples,
   SampleReference64Ptr sampleRefs,
   TimeValue *sampleTime
);
```

Parameters

theMedia
The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (page 1630) and `GetTrackMedia` (page 1612). See `Media Identifiers`.

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 sound description structure. If you don't want the `SampleDescription` structure, set this parameter to `NIL`.

numberOfSamples
The number of `SampleReference64Record` structures pointed to by the `sampleRefs` parameter. Each structure may contain one or more contiguous samples. For details, see `AddMediaSample` (page 1536).

sampleRefs

A pointer to the number of `SampleReference64Record` structures specified by the `numberOfSamples` parameter.

sampleTime

A pointer to a time value. After adding the reference to the media, the `AddMediaSampleReferences` function returns the time where the reference was inserted, using the time scale referred to by this parameter. If you don't want to receive this information, set this parameter to `NIL`.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

The only difference between this function and `AddMediaSampleReferences` (page 1543) is that the `sampleRefs` parameter points to SampleReference64Record structures instead of `SampleReferenceRecord` structures.

Special Considerations

New applications should use this function instead of the 32-bit version.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CreateMovieFromReferences

Declared In

`Movies.h`

AddMovieSelection

Adds one or more tracks to a movie.

```
void AddMovieSelection (
   Movie theMovie,
   Movie src
);
```

Parameters

theMovie

The destination movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

src

The source movie for this operation. `AddMovieSelection` adds the tracks from this movie to the destination movie. The function adds these tracks at the time specified by the current selection in the destination movie.

Return Value

You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Discussion

This function scales the source movie so that it fits into the destination selection. If the current selection in the destination movie has a 0 duration, the Movie Toolbox adds the segment at the beginning of the current selection. The entire source movie is used regardless of the selection in the source movie. The Movie Toolbox removes any empty tracks from the destination movie after the add operation. If you have assigned a progress function to the destination movie, the Movie Toolbox calls that progress function during long add operations. Following is an example of using this function:

```
// AddMovieSelection coding example
// See "Discovering QuickTime," page 363
Movie           movie1;
TimeValue       lOldDuration;
Movie           movie2;
long            lIndex, lOrigTrackCount, lReferenceIndex;
Track           track, trackSprite;
// get the first track in original movie and position at the start
trackSprite =GetMovieIndTrack(movie1, 1);
SetMovieSelection(movie1, 0, 0);
// remove all tracks except video in modifier movie
for (lIndex =1; lIndex <=GetMovieTrackCount(movie2); lIndex++) {
    Track       track =GetMovieIndTrack(movie2, lIndex);
    OSType      dwType;
    GetMediaHandlerDescription(GetTrackMedia(track),
                                &dwType, NIL, NIL);
    if (dwType !=VideoMediaType) {
        DisposeMovieTrack(track);
        lIndex--;
    }
}
// add the modifier track to original movie
lOldDuration =GetMovieDuration(movie1);
AddMovieSelection(movie1, movie2);
DisposeMovie(movie2);
// truncate the movie to the length of the original track
DeleteMovieSegment(movie1, lOldDuration,
                    GetMovieDuration(movie1) - lOldDuration);
// associate the modifier track with the original sprite track
track =GetMovieIndTrack(movie1, lOrigTrackCount + 1);
AddTrackReference(trackSprite, track, kTrackModifierReference,
                    &lReferenceIndex);
```

Special Considerations

Some Movie Toolbox functions can take a long time to execute. For example, if you call `FlattenMovie` (page 1336) and specify a large movie, the Movie Toolbox must read and write all the sample data for the movie. During such operations you may wish to display some kind of progress indicator to the user. A progress function is an application-defined function that you can create to track the progress of time-consuming activities and keep the user informed.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

SlideShowImporter

SlideShowImporter.win

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` (page 1630) and `GetTrackMedia` (page 1612).

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` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result.

Discussion
This function can return these errors:

Availability
Available in Mac OS X v10.3 and later.

Declared In
`Movies.h`

AddTrackReference

Adds a new track reference to a track.

```
OSErr AddTrackReference (
   Track theTrack,
   Track refTrack,
   OSType refType,
   long *addedIndex
);
```

Parameters

theTrack

Identifies the track for this operation. Your application obtains this track identifier from such functions as NewMovieTrack (page 1628) and GetMovieTrack (page 1601).

refTrack

The track to be identified in the track reference.

refType

The type of reference.

addedIndex

A pointer to a long integer. The toolbox returns the index value assigned to the new track reference. If you don't want this information, set this parameter to NIL.

Return Value

You can access Movie Toolbox error returns through GetMoviesError (page 221) and GetMoviesStickyError (page 222), as well as in the function result. See Error Codes.

Discussion

The following code snippet shows how AddTrackReference can be used to add a modifier track reference to a sprite track.

```
// AddTrackReference coding example
// See "Discovering QuickTime," page 363
Movie           movie1;
TimeValue       lOldDuration;
Movie           movie2;
long            lIndex, lOrigTrackCount, lReferenceIndex;
Track           track, trackSprite;
// get the first track in original movie and position at the start
trackSprite =GetMovieIndTrack(movie1, 1);
SetMovieSelection(movie1, 0, 0);
// remove all tracks except video in modifier movie
for (lIndex =1; lIndex <=GetMovieTrackCount(movie2); lIndex++) {
    Track       track =GetMovieIndTrack(movie2, lIndex);
    OSType      dwType;
    GetMediaHandlerDescription(GetTrackMedia(track),
                                &dwType, NIL, NIL);
    if (dwType !=VideoMediaType) {
        DisposeMovieTrack(track);
        lIndex--;
    }
}
// add the modifier track to original movie
lOldDuration =GetMovieDuration(movie1);
AddMovieSelection(movie1, movie2);
DisposeMovie(movie2);
// truncate the movie to the length of the original track
DeleteMovieSegment(movie1, lOldDuration,
                    GetMovieDuration(movie1) - lOldDuration);
// associate the modifier track with the original sprite track
```

```
track =GetMovieIndTrack(movie1, lOrigTrackCount + 1);
AddTrackReference(trackSprite, track, kTrackModifierReference,
                    &lReferenceIndex);
```

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
makeeffectslideshow

samplemakeeffectmovie.win

vrmakepano

VRMakePano Library

vrmakepano.win

Declared In
`Movies.h`

BeginMediaEdits

Starts a media-editing session.

```
OSErr BeginMediaEdits (
   Media theMedia
);
```

Parameters

`theMedia`
The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (page 1630) and `GetTrackMedia` (page 1612). See `Media Identifiers`.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
Use `EndMediaEdits` (page 1567) to end a media-editing session. You must call `BeginMediaEdits` before you add samples to a media with the `AddMediaSample` (page 1536) function. You must also call `BeginMediaEdits` before calling `InsertTrackSegment` (page 1623) if you wish `InsertTrackSegment` to copy media samples instead of copying the segment by reference.

```
// BeginMediaEdits coding example
// See "Discovering QuickTime," page 89
void CreateMyVideoTrack (Movie movie)
{
    Track   track;
    Media   media;
    Rect    rect ={0, 0, 100, 320};
    track =NewMovieTrack(movie,
                FixRatio(rect.right, 1),
                FixRatio(rect.bottom, 1),
                kNoVolume);
    media =NewTrackMedia(track,
                VideoMediaType,
                600,                            // video time scale
                NIL, NIL);
    BeginMediaEdits(media);
    MyAddVideoSamplesToMedia(media, &rect);     // assemble data
    EndMediaEdits(media);
    InsertMediaIntoTrack(track,
                0,                              // track start time
                0,                              // media start time
                GetMediaDuration(media),
                kFix1);                         // normal speed
}
```

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qteffects
qteffects.win
vrmakepano
VRMakePano Library
vrmakepano.win

Declared In
`Movies.h`

ClearMovieSelection

Removes the segment of the movie that is defined by the current selection.

```
void ClearMovieSelection (
   Movie theMovie
);
```

Parameters

theMovie
    The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

ConvertDataRefToMovieDataRef

Converts a piece of data in a storage location to a movie file format and stores it in another storage location, supporting a user settings dialog box for import operations.

```
OSErr ConvertDataRefToMovieDataRef (
   Handle inputDataRef,
   OSType inputDataRefType,
   Handle outputDataRef,
   OSType outputDataRefType,
   OSType creator,
   long flags,
   ComponentInstance userComp,
   MovieProgressUPP proc,
   long refCon
);
```

Parameters

inputDataRef
A data reference that specifies the storage location of the source data.

inputDataRefType
The type of the input data reference.

outputDataRef
A data reference that specified the storage location to receive the converted data.

outputDataRefType
The type of the output data reference.

creator
The creator type of the output storage location.

flags
Flags (see below) that control the operation of the dialog box. See these constants:
`createMovieFileDeleteCurFile`
`movieToFileOnlyExport`
`movieFileSpecValid`
`showUserSettingsDialog`

userComp
An instance of a component to be used for converting the movie data.

proc
A progress callback function; see `MovieProgressProc` in the QuickTime API Reference.

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

Return Value
See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error.

Discussion
This function converts a piece of data in a storage location into a movie and stores into another storage location. Both the input and the output storage locations are specified through data references. If the storage location is on a local file system, the file will have the specified creator. If specified as such in the flags, the function displays a dialog box that lets the user to choose the output file and the export type. If an export component (or its instance) is specified in `userComp`, it will be used for the conversion operation.

Version Notes
Introduced in QuickTime 6.4.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`Movies.h`

ConvertFileToMovieFile

Converts a file to a movie file and supports a user settings dialog box for import operations.

```
OSErr ConvertFileToMovieFile (
   const FSSpec *inputFile,
   const FSSpec *outputFile,
   OSType creator,
   ScriptCode scriptTag,
   short *resID,
   long flags,
   ComponentInstance userComp,
   MovieProgressUPP proc,
   long refCon
);
```

Parameters

inputFile
A pointer to the file system specification for the file to be converted into a movie file.

outputFile
A pointer to the file specification for the destination movie file.

creator
The creator value for the file if it is a new one.

scriptTag
The script in which the movie file should be converted. Use the Script Manager constant `smSystemScript` to use the system script; use the `smCurrentScript` constant to use the current script. See *Inside Macintosh: Text* for more information about scripts and script tags.

resID
A pointer to a field that is to receive the resource ID of the file to be converted. If you don't want to receive the resource ID, set this parameter to `NIL`.

flags
Contains flags (see below) that control movie file conversion and determine whether or not the user settings dialog box appears. See these constants:

`createMovieFileDeleteCurFile`

`movieToFileOnlyExport`

`movieFileSpecValid`

`showUserSettingsDialog`

userComp

Indicates a component or component instance of the movie export component you want to perform the conversion. Otherwise, set this parameter to 0 for the Movie Toolbox to choose the appropriate component. If you pass in a component instance, it will be used by `ConvertFileToMovieFile`. This allows you to communicate directly with the component before using this function to establish any conversion parameters. If you pass in a component ID, an instance is created and closed within this function.

proc

Points to your progress callback. To remove a movie's progress function, set this parameter to `NIL`. Set this parameter to -1 for the Movie Toolbox to provide a default progress function. See `MovieProgressProc` for the interface your progress callback must support.

refCon

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

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

Use this function to specify an input file and convert it to a movie file. Because some conversions may take a nontrivial amount of time, you can pass a standard movie progress function in the `proc` and `refCon` parameters.

Special Considerations

Once you are finished working with a movie, you should release the resources used by the movie by calling `DisposeMovie` (page 188).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

ImportExportMovie

qtdataexchange

qtdataexchange.win

Declared In

`Movies.h`

ConvertMovieToDataRef

Converts a specified movie (or a single track within a movie) into a specified file format and stores it in a specified storage location.

```
OSErr ConvertMovieToDataRef (
   Movie m,
   Track onlyTrack,
   Handle dataRef,
   OSType dataRefType,
   OSType fileType,
   OSType creator,
   long flags,
   ComponentInstance userComp
);
```

Parameters

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

onlyTrack
The track in the source movie, if you want to convert only a single track.

dataRef
A data reference that specifies the storage location to receive the converted movie data.

dataRefType
The type of data reference. This function currently supports only alias data references.

fileType
The Mac OS file type of the storage location, which determines the export format.

creator
The creator type of the storage location.

flags
Flags (see below) that control the operation of the dialog box. See these constants:
`showUserSettingsDialog`
`movieToFileOnlyExport`
`movieFileSpecValid`

userComp
An instance of the component to be used for converting the movie data.

Return Value
See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error.

Discussion
If the storage location is on a local file system, the file will have the specified file type and the creator. If specified as such in the flags, the function displays a dialog box that lets the user choose the output file and the export type. If an export component (or its instance) is specified in the `userComp` parameter, it will be used to perform the conversion operation.

Version Notes
Introduced in QuickTime 6.4.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
BackgroundExporter

Declared In

`Movies.h`

ConvertMovieToFile

Takes a specified movie (or a single track within that movie) and converts it into a specified file and type, supporting a Save As dialog box.

```
OSErr ConvertMovieToFile (
   Movie theMovie,
   Track onlyTrack,
   FSSpec *outputFile,
   OSType fileType,
   OSType creator,
   ScriptCode scriptTag,
   short *resID,
   long flags,
   ComponentInstance userComp
);
```

Parameters

theMovie

The source movie for this conversion operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

onlyTrack

The track within the source movie for this conversion operation. To specify all tracks, set the `value` of this parameter to 0.

outputFile

A pointer to the file specification for the destination file.

fileType

The data type of the destination file for the movie specified in the parameter `theMovie`.

creator

The creator value for the output file if it is a new one.

scriptTag

The script into which the movie should be converted if the output file is a new one. Use the Script Manager constant `smSystemScript` to use the system script; use the `smCurrentScript` constant to use the current script. See *Inside Macintosh: Text* for more information about scripts and script tags.

resID

A pointer to a field that is to receive the resource ID of the open movie. If you don't want to receive this information, set the `resID` parameter to `NIL`.

flags

Contains flags (see below) that control whether and how the Save As dialog box appears. See these constants:

`showUserSettingsDialog`

`movieToFileOnlyExport`

`movieFileSpecValid`

userComp

If you want a particular movie export component to perform the conversion, you may pass the component or an instance of that component in this parameter. Otherwise, set it to 0 to allow the Movie Toolbox to use the appropriate component. If you pass in a component instance, it is used by `ConvertMovieToFile`. This allows you to communicate directly with the component before making this call to establish any conversion parameters. If you pass in a component ID, an instance is created and closed within this call.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

Your application controls whether a Save As dialog box appears by setting the value of the `flags` parameter. The dialog box lets the user specify the file name and type. Supported types include standard QuickTime movies, flattened movies, single-fork flattened movies, and any format that is supported by a movie data export component. The following code snippets show how to call `ConvertMovieToFile` to provide a simple export capability and how to save a sound-only QuickTime movie as a WAV file.

```
// Providing an export capability with ConvertMovieToFile
err =ConvertMovieToFile (theMovie,     /* identifies movie */
                    NIL,                /* all tracks */
                    NIL,                /* no output file */
                    0,                  /* no file type */
                    0,                  /* no creator */
                    -1,                 /* script */
                    NIL,                /* no resource ID */
                    createMovieFileDeleteCurFile |
                        showUserSettingsDialog |
                        movieToFileOnlyExport,
                    0);                 /* no specific component */
// Saving a sound-only QuickTime movie as a WAVE file
// See "Discovering QuickTime," page 257
void SndSnip_SaveSoundMovieAsWAVEFile (Movie theMovie)
{
    StandardFileReply   myReply;
    // have the user select the name and location of the new WAVE file
    StandardPutFile("\pSave sound movie file as:",
                                        "\pUntitled.wav", &myReply);
    if (!myReply.sfGood)
        return;
    // use the default progress procedure, if any
    SetMovieProgressProc(theMovie, (MovieProgressUPP)-1L, 0);
    // export the movie into a file
    ConvertMovieToFile( theMovie,               // the movie to convert
                        NIL,                   // all tracks in the movie
                        &myReply.sfFile,        // the output file
                        kQTFileTypeWave,        // the output file type
                        FOUR_CHAR_CODE('TVOD'), // the output file creator
                        smSystemScript,         // the script
                        NIL,                   // no resource ID
                                                //   to be returned
                        0L,                     // no flags
                        NIL);                  // no specific component
}
```

Version Notes

Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
ImportExportMovie
MovieToAIFF
qthintmovies.win
soundsnippets
soundsnippets.win

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` (page 1630) and `GetTrackMedia` (page 1612).

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` (page 1269).

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` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result.

Discussion
To find out how many samples were returned in the sample table, call `QTSampleTableGetNumberOfSamples` (page 1265).

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
MovieVideoChart

Declared In
`Movies.h`

CopyMovieSelection

Creates a new movie that contains the original movie's current selection.

```
Movie CopyMovieSelection (
   Movie theMovie
);
```

Parameters

theMovie
: The source movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

Return Value
The new movie.

Discussion
This function creates a new movie from the source movie's current selection, but does not change the source movie or the selection. If you have assigned a progress function to the source movie, the Movie Toolbox calls that progress function during long copy operations.

Special Considerations
Your application must dispose of the new movie once you are done with it, using `DisposeMovie` (page 188).

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

CopyMovieSettings

Copies many settings from one movie to another, overwriting the destination settings in the process.

```
OSErr CopyMovieSettings (
   Movie srcMovie,
   Movie dstMovie
);
```

Parameters

srcMovie
The source movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

dstMovie
The destination movie for this operation. The `CopyMovieSettings` function uses the settings from the source movie, which is specified by the `srcMovie` parameter, to replace the current settings of this movie.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
Use this function to copy certain important settings from one movie to another. It copies the preferred rate and volume, source clipping region, matrix information, and user data; it does not copy the movie's contents. To work with movie contents, you should use segment editing functions.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
MakeEffectMovie
qtcompress.win
qteffects
qteffects.win
samplemakeeffectmovie.win

Declared In
`Movies.h`

CopyTrackSettings

Copies many settings from one track to another, overwriting the destination settings.

```
OSErr CopyTrackSettings (
   Track srcTrack,
   Track dstTrack
);
```

Parameters

srcTrack
The source track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

dstTrack
: The destination track for this operation. The `CopyTrackSettings` function uses the settings from the source track, which you specify with the `srcTrack` parameter, to replace the current settings of this track.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
This function copies matrix information, track volume, the clipping region, user data, matte information, media language, quality, user data, and other media-specific settings (such as sound balance and video graphics mode). It does not copy any alternate group information pertaining to the track. This function does not copy the track's contents. To work with track contents, you should use segment-editing functions.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
BurntTextSampleCode

qtaddeffectseg

qtaddeffectseg.win

qteffects

qteffects.win

Declared In
`Movies.h`

CutMovieSelection

Creates a new movie that contains the original movie's current selection.

```
Movie CutMovieSelection (
   Movie theMovie
);
```

Parameters

theMovie
: The source movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

Return Value
The newly created movie.

Discussion
This function removes the current selection from the original movie and makes the selection into a new movie. After the current selection has been removed from the original movie, the duration of the current selection is 0. The starting time of the current selection is not affected. If you have assigned a progress function to the source movie, the Movie Toolbox calls that progress function during long cut operations.

Special Considerations

Your application must dispose of the new movie once you are done with it, using DisposeMovie (page 188).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

Movies.h

DeleteMovieSegment

Removes a specified segment from a movie.

```
OSErr DeleteMovieSegment (
   Movie theMovie,
   TimeValue startTime,
   TimeValue duration
);
```

Parameters

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as NewMovie (page 259), NewMovieFromFile (page 1398), and NewMovieFromHandle (page 1400).

startTime

A time value specifying the starting point of the segment to be deleted.

duration

A time value that specifies the duration of the segment to be deleted.

Return Value

You can access Movie Toolbox error returns through GetMoviesError (page 221) and GetMoviesStickyError (page 222), as well as in the function result. See Error Codes.

Discussion

You identify the segment to remove by specifying its starting time and duration. The following code snippet shows DeleteMovieSegment being used while adding a modifier track to a movie.

```
// DeleteMovieSegment coding example
// See "Discovering QuickTime," page 363
Movie           movie1;
TimeValue       lOldDuration;
Movie           movie2;
long            lIndex, lOrigTrackCount, lReferenceIndex;
Track           track, trackSprite;
// get the first track in original movie and position at the start
trackSprite =GetMovieIndTrack(movie1, 1);
SetMovieSelection(movie1, 0, 0);
// remove all tracks except video in modifier movie
for (lIndex =1; lIndex <=GetMovieTrackCount(movie2); lIndex++) {
    Track       track =GetMovieIndTrack(movie2, lIndex);
    OSType      dwType;
    GetMediaHandlerDescription(GetTrackMedia(track),
                                &dwType, NIL, NIL);
    if (dwType !=VideoMediaType) {
        DisposeMovieTrack(track);
        lIndex--;
    }
}
// add the modifier track to original movie
lOldDuration =GetMovieDuration(movie1);
AddMovieSelection(movie1, movie2);
DisposeMovie(movie2);
// truncate the movie to the length of the original track
DeleteMovieSegment(movie1, lOldDuration,
                    GetMovieDuration(movie1) - lOldDuration);
// associate the modifier track with the original sprite track
track =GetMovieIndTrack(movie1, lOrigTrackCount + 1);
AddTrackReference(trackSprite, track, kTrackModifierReference,
                    &lReferenceIndex);
```

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qtspritesplus.win

Declared In
`Movies.h`

DeleteTrackReference

Removes a track reference from a track.

```
OSErr DeleteTrackReference (
   Track theTrack,
   OSType refType,
   long index
);
```

Parameters

theTrack
Identifies the track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

refType
The type of reference.

index
The index value of the reference to be deleted. You obtain this index value when you create the track reference.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

This function deletes a track reference from a track. If there are additional track references with higher index values, the toolbox automatically renumbers those references, decrementing their index values by 1.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

MakeEffectMovie

qteffects.win

QTKitTimeCode

qttext

qttext.win

Declared In

`Movies.h`

DeleteTrackSegment

Removes a specified segment from a track.

```
OSErr DeleteTrackSegment (
   Track theTrack,
   TimeValue startTime,
   TimeValue duration
);
```

Parameters

`theTrack`

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

`startTime`

A time value specifying the starting point of the segment to be deleted. This time value must be expressed in the time scale of the movie that contains the source track.

`duration`

A time value that specifies the duration of the segment to be deleted. This time value must be expressed in the time scale of the movie that contains the source track.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

You identify the segment to remove by specifying its starting time and duration.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

addflashactions.win

addhtactions.win

qttext.win

qtwiredactions

qtwiredactions.win

Declared In

`Movies.h`

DisposeMovieEditState

Disposes of an edit state.

```
OSErr DisposeMovieEditState (
   MovieEditState state
);
```

Parameters

`state`

The edit state for this operation. Your application obtains this edit state identifier when you create the edit state by calling `NewMovieEditState` (page 1628).

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Special Considerations

You must dispose of a movie's edit states before you dispose of the movie itself.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

DisposeMovieTrack

Removes a track from a movie.

```
void DisposeMovieTrack (
   Track theTrack
);
```

Parameters

`theTrack`

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

Return Value

You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Discussion

The following code snippet illustrates the use of `DisposeMovieTrack`:

```
// DisposeMovieTrack coding example
// See "Discovering QuickTime," page 363
Movie           movie1;
TimeValue       lOldDuration;
Movie           movie2;
long            lIndex, lOrigTrackCount, lReferenceIndex;
Track           track, trackSprite;
// get the first track in original movie and position at the start
trackSprite =GetMovieIndTrack(movie1, 1);
SetMovieSelection(movie1, 0, 0);
// remove all tracks except video in modifier movie
for (lIndex =1; lIndex <=GetMovieTrackCount(movie2); lIndex++) {
    Track       track =GetMovieIndTrack(movie2, lIndex);
    OSType      dwType;
    GetMediaHandlerDescription(GetTrackMedia(track),
                                &dwType, NIL, NIL);
    if (dwType !=VideoMediaType) {
        DisposeMovieTrack(track);
        lIndex--;
    }
}
// add the modifier track to original movie
lOldDuration =GetMovieDuration(movie1);
AddMovieSelection(movie1, movie2);
DisposeMovie(movie2);
// truncate the movie to the length of the original track
DeleteMovieSegment(movie1, lOldDuration,
                    GetMovieDuration(movie1) - lOldDuration);
// associate the modifier track with the original sprite track
track =GetMovieIndTrack(movie1, lOrigTrackCount + 1);
AddTrackReference(trackSprite, track, kTrackModifierReference,
                    &lReferenceIndex);
```

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qtinfo

QTKitTimeCode

qttext

qttext.win

qttimecode.win

Declared In

`Movies.h`

DisposeTrackEditState

Disposes of a movie's track edit state.

```
OSErr DisposeTrackEditState (
   TrackEditState state
);
```

Parameters

state

The edit state for this operation. Your application obtains this edit state identifier when you create the edit state by calling the `NewTrackEditState` (page 1630) function.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

Your application must dispose of any edit states you create. You create an edit state by calling `NewTrackEditState` (page 1630).

Special Considerations

You must dispose of a movie's track edit states before you dispose of the track or the movie.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

DisposeTrackMedia

Removes a media from a track.

```
void DisposeTrackMedia (
   Media theMedia
);
```

Parameters

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (page 1630) and `GetTrackMedia` (page 1612). See `Media Identifiers`.

Return Value

You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Discussion

This function does not remove the track from its movie.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTKitTimeCode

Declared In
`Movies.h`

EndMediaEdits

Ends a media-editing session.

```
OSErr EndMediaEdits (
   Media theMedia
);
```

Parameters

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (page 1630) and `GetTrackMedia` (page 1612). See `Media Identifiers`.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
The following code sample illustrates the use of `EndMediaEdits`:

```
// EndMediaEdits coding example
// See "Discovering QuickTime," page 89
void CreateMyVideoTrack (Movie movie)
{
    Track   track;
    Media   media;
    Rect    rect ={0, 0, 100, 320};
    track =NewMovieTrack(movie,
                FixRatio(rect.right, 1),
                FixRatio(rect.bottom, 1),
                kNoVolume);
    media =NewTrackMedia(track,
                VideoMediaType,
                600,                            // video time scale
                NIL, NIL);
    BeginMediaEdits(media);
    MyAddVideoSamplesToMedia(media, &rect);     // assemble data
    EndMediaEdits(media);
    InsertMediaIntoTrack(track,
                0,                              // track start time
                0,                              // media start time
                GetMediaDuration(media),
                kFix1);                         // normal speed
}
```

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qteffects

qteffects.win

vrmakepano

VRMakePano Library

vrmakepano.win

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` (page 1630) and `GetTrackMedia` (page 1612).

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` (page 221) and `GetMoviesStickyError` (page 222), 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).

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
CaptureAndCompressIPBMovie

OpenGLCaptureToMovie

Quartz Composer QCTV

Declared In
`Movies.h`

GetDataHandler

Retrieves the best data handler component to use with a given data reference.

```
Component GetDataHandler (
   Handle dataRef,
   OSType dataHandlerSubType,
   long flags
);
```

Parameters

dataRef

A handle to the `data` reference. The type of information stored in the handle depends upon the `data` reference type specified by the `dataHandlerSubType` parameter.

dataHandlerSubType

Identifies both the type of data reference and, by implication, the `component` subtype value assigned to the data handler components that operate on data references of that type.

flags

Contains flags (see below) that indicate the way in which you intend to use the data handler component. Note that not all data handlers necessarily support all services; for example, some data handler components may not support streaming writes. Set the appropriate flags to 1. See these constants:

`kDataHCanRead`

`kDataHCanWrite`

`kDataHCanStreamingWrite`

Return Value

The best data handler component conforming to the parameters passed in.

Discussion

Once you have used this function to get information about the best data handler component for your data reference, you can open and use the component using Component Manager functions. If the function returns a value of `NIL`, the toolbox was unable to find an appropriate data handler component. For more information about the error that caused a return of `NIL`, call `GetMoviesError` (page 221).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

ElectricImageComponent.win

qtdataref

qtfiletransfer

qtfiletransfer.win

ThreadsImportMovie

Declared In

`Movies.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` (page 1630) and `GetTrackMedia` (page 1612).

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.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

MovieVideoChart

Declared In

`Movies.h`

GetMediaCreationTime

Returns the creation date and time stored in a media.

```
unsigned long GetMediaCreationTime (
   Media theMedia
);
```

Parameters

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (page 1630) and `GetTrackMedia` (page 1612). See `Media Identifiers`.

Return Value

The media's creation date and time.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

GetMediaDataHandler

Determines a media's data handler.

```
DataHandler GetMediaDataHandler (
   Media theMedia,
   short index
);
```

Parameters

theMedia

The media for this operation. Your application obtains this media identifier from such functions as NewTrackMedia (page 1630) and GetTrackMedia (page 1612). See Media Identifiers.

index

Identifies the data reference. You provide the index value that corresponds to the data reference for which you want to retrieve the data handler. You must set this parameter to 1.

Return Value

A data handler component instance.

Special Considerations

QuickTime normally takes care of selecting data handlers for media. Your application should not need to call this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

Movies.h

GetMediaDataHandlerDescription

Retrieves information about a media's data handler.

```
void GetMediaDataHandlerDescription (
   Media theMedia,
   short index,
   OSType *dhType,
   Str255 creatorName,
   OSType *creatorManufacturer
);
```

Parameters

theMedia

The media for this operation. Your application obtains this media identifier from such functions as NewTrackMedia (page 1630) and GetTrackMedia (page 1612). See Media Identifiers.

index

Identifies the data reference. You provide the index value that corresponds to the data reference for which you want to retrieve the data handler description. You must set this parameter to 1.

dhType

A pointer to a field of data type `OSType`. The Movie Toolbox returns the `data` handler type identifier. This value indicates the type of data reference supported by this data handler. This value also corresponds to the `component` subtype specified for the data handler component. All QuickTime data references have a type value of `'alis'`. If you don't want to receive this information, set this parameter to `NIL`.

creatorName

Points to a string. The Movie Toolbox returns the name of the data handler's creator. If you don't want to receive this information, set this parameter to `NIL`.

creatorManufacturer

A pointer to a long integer. The Movie Toolbox returns the 4-byte value that identifies the manufacturer of the component. If you don't want to retrieve this information, set this parameter to `NIL`.

Return Value

You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

GetMediaDataSize

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

```
long GetMediaDataSize (
   Media theMedia,
   TimeValue startTime,
   TimeValue duration
);
```

Parameters

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (page 1630) and `GetTrackMedia` (page 1612). See `Media Identifiers`.

startTime

A time value specifying the starting point of the segment.

duration

A time value that specifies the duration of the segment.

Return Value

The size, in bytes, of the sample data in the defined media segment.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

GetMediaDataSize64

Provides a 64-bit version of GetMediaDataSize.

```
OSErr GetMediaDataSize64 (
   Media theMedia,
   TimeValue startTime,
   TimeValue duration,
   wide *dataSize
);
```

Parameters

theMedia
: The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (page 1630) and `GetTrackMedia` (page 1612). See `Media Identifiers`.

startTime
: A time value specifying the starting point of the segment.

duration
: A time value that specifies the duration of the segment.

dataSize
: The size, in bytes, of the sample data in the defined media segment.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
The only difference between this function and `GetMediaDataSize` (page 1572) is that the `dataSize` parameter returns a 64-bit integer instead of the function returning a 32-bit integer.

Special Considerations

New applications should use this function instead of the 32-bit version.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

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` (page 1630) and `GetTrackMedia` (page 1612).

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` (page 221) and `GetMoviesStickyError` (page 222), 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.

Availability

Available in Mac OS X v10.3 and later.

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` (page 1630) and `GetTrackMedia` (page 1612).

Return Value

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

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

MovieVideoChart

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` (page 1630) and `GetTrackMedia` (page 1612).

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:

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

CaptureAndCompressIPBMovie

MovieVideoChart

OpenGLCaptureToMovie

Quartz Composer QCTV

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` (page 1630) and `GetTrackMedia` (page 1612).

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.

Availability
Available in Mac OS X v10.3 and later.

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` (page 1630) and `GetTrackMedia` (page 1612).

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.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`Movies.h`

GetMediaDuration

Returns the duration of a media.

```
TimeValue GetMediaDuration (
   Media theMedia
);
```

Parameters

`theMedia`
The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (page 1630) and `GetTrackMedia` (page 1612). See `Media Identifiers`.

Return Value
The media's duration.

Discussion
The following code sample illustrates the use of `GetMediaDuration`:

```
// GetMediaDuration coding example
// See "Discovering QuickTime," page 89
void CreateMyVideoTrack (Movie movie)
{
    Track   track;
    Media   media;
    Rect    rect ={0, 0, 100, 320};
    track =NewMovieTrack(movie,
                FixRatio(rect.right, 1),
                FixRatio(rect.bottom, 1),
                kNoVolume);
    media =NewTrackMedia(track,
                VideoMediaType,
                600,                            // video time scale
                NIL, NIL);
    BeginMediaEdits(media);
    MyAddVideoSamplesToMedia(media, &rect);     // assemble data
    EndMediaEdits(media);
    InsertMediaIntoTrack(track,
                0,                              // track start time
                0,                              // media start time
                GetMediaDuration(media),
                kFix1);                         // normal speed
}
```

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qteffects

qteffects.win

qtshoweffect

vrmakepano

VRMakePano Library

Declared In
`Movies.h`

GetMediaHandler

Obtains a reference to a media handler component.

```
MediaHandler GetMediaHandler (
   Media theMedia
);
```

Parameters

`theMedia`
: The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (page 1630) and `GetTrackMedia` (page 1612). See `Media Identifiers`.

Return Value
A media handler component instance.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qttext
qttext.win
qtwiredactions
vrscript
vrscript.win

Declared In
`Movies.h`

GetMediaHandlerDescription

Retrieves information about a media handler.

```
void GetMediaHandlerDescription (
   Media theMedia,
   OSType *mediaType,
   Str255 creatorName,
   OSType *creatorManufacturer
);
```

Parameters

theMedia
The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (page 1630) and `GetTrackMedia` (page 1612). See `Media Identifiers`.

mediaType
A pointer to a field in which the Movie Toolbox returns the `media` type identifier (see below). This value indicates the type of media supported by this media handler. This value also corresponds to the `component` subtype specified for the media handler component. If you don't want to receive this information, set the `mediaType` parameter to `NIL`. See these constants:
`VideoMediaType`
`SoundMediaType`
`TextMediaType`

creatorName
Points to a string. The Movie Toolbox returns the name of the media handler's creator. If you don't want to receive this information, set this parameter to `NIL`.

creatorManufacturer
A pointer to a long integer. The Movie Toolbox returns the 4-byte value that identifies the manufacturer of the component. If you don't want to retrieve this information, set this parameter to `NIL`.

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Discussion
The following code sample illustrates the use of `GetMediaHandlerDescription`:

```
// GetMediaHandlerDescription coding example
// See "Discovering QuickTime," page 363
Movie           movie1;
TimeValue       lOldDuration;
Movie           movie2;
long            lIndex, lOrigTrackCount, lReferenceIndex;
Track           track, trackSprite;
// get the first track in original movie and position at the start
trackSprite =GetMovieIndTrack(movie1, 1);
SetMovieSelection(movie1, 0, 0);
// remove all tracks except video in modifier movie
for (lIndex =1; lIndex <=GetMovieTrackCount(movie2); lIndex++) {
    Track       track =GetMovieIndTrack(movie2, lIndex);
    OSType      dwType;
    GetMediaHandlerDescription(GetTrackMedia(track),
                                &dwType, NIL, NIL);
    if (dwType !=VideoMediaType) {
        DisposeMovieTrack(track);
        lIndex--;
    }
}
// add the modifier track to original movie
lOldDuration =GetMovieDuration(movie1);
AddMovieSelection(movie1, movie2);
DisposeMovie(movie2);
// truncate the movie to the length of the original track
DeleteMovieSegment(movie1, lOldDuration,
                    GetMovieDuration(movie1) - lOldDuration);
// associate the modifier track with the original sprite track
track =GetMovieIndTrack(movie1, lOrigTrackCount + 1);
AddTrackReference(trackSprite, track, kTrackModifierReference,
                    &lReferenceIndex);
```

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CompressMovies
DigitizerShell
DragAndDrop Shell
MovieGWorlds
QT Internals

Declared In
`Movies.h`

GetMediaInputMap

Returns a copy of the input map associated with a specified media.

```
OSErr GetMediaInputMap (
   Media theMedia,
   QTAtomContainer *inputMap
);
```

Parameters

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (page 1630) and `GetTrackMedia` (page 1612). See `Media Identifiers`.

inputMap

The media input map for this operation. You must dispose of the map referred to by this parameter when you are done with it using `QTDisposeAtomContainer` (page 1427).

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

Use this function to specify the media you want to get so you can modify its input map, as illustrated below:

```
// GetMediaInputMap coding example
// See "Discovering QuickTime," page 365
#define kImageIndexToOverride          1
Movie                   movie1, movie2;
long                    lReferenceIndex, lImageIndexToOverride;
Track                   trackSprite;
QTAtomContainer         qtacInputMap;
QTAtom                  lInputAtom;
OSType                  dwInputType;
Media                   mediaSprite;
// get the sprite media's input map
mediaSprite =GetTrackMedia(trackSprite);
GetMediaInputMap(mediaSprite, &qtacInputMap);
// add an atom for a modifier track
QTInsertChild(qtacInputMap, kParentAtomIsContainer, kTrackModifierInput,
                lReferenceIndex, 0, 0, NIL, &lInputAtom);
// add a child atom to specify the input type
dwInputType =kTrackModifierTypeImage;
QTInsertChild(qtacInputMap, lInputAtom, kTrackModifierType, 1, 0,
                sizeof(dwInputType), &dwInputType, NIL);
// add a second child atom to specify index of image to override
lImageIndexToOverride =EndianS16_NtoB(kImageIndexToOverride);
QTInsertChild(qtacInputMap, lInputAtom, kSpritePropertyImageIndex, 1, 0,
        sizeof(lImageIndexToOverride), &lImageIndexToOverride, NIL);
// update the sprite media's input map
SetMediaInputMap(mediaSprite, qtacInputMap);
QTDisposeAtomContainer(qtacInputMap);
```

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

ChromaKeyMovie

Declared In
`Movies.h`

GetMediaLanguage

Returns a media's localized language or region code.

```
short GetMediaLanguage (
   Media theMedia
);
```

Parameters

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (page 1630) and `GetTrackMedia` (page 1612). See `Media Identifiers`.

Return Value
The media's language or region code.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

GetMediaModificationTime

Returns a media's modification date and time.

```
unsigned long GetMediaModificationTime (
   Media theMedia
);
```

Parameters

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (page 1630) and `GetTrackMedia` (page 1612). See `Media Identifiers`.

Return Value
The media's modification date and time.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

GetMediaPreferredChunkSize

Retrieves the maximum chunk size for a media.

```
OSErr GetMediaPreferredChunkSize (
   Media theMedia,
   long *maxChunkSize
);
```

Parameters

theMedia

The media for this operation. Your application obtains this media identifier from such functions as NewTrackMedia (page 1630) and GetTrackMedia (page 1612). See Media Identifiers.

maxChunkSize

Specifies a field to receive the maximum chunk size, in bytes.

Return Value

You can access Movie Toolbox error returns through GetMoviesError (page 221) and GetMoviesStickyError (page 222), as well as in the function result. See Error Codes.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

Movies.h

GetMediaQuality

Returns a media's quality level value.

```
short GetMediaQuality (
   Media theMedia
);
```

Parameters

theMedia

The media for this operation. Your application obtains this media identifier from such functions as NewTrackMedia (page 1630) and GetTrackMedia (page 1612). See Media Identifiers.

Return Value

A short integer whose bits indicate quality constants (see below). More than one of these bits may be set to 1.

Discussion

The Movie Toolbox uses this quality value to influence which track of a movie it selects to play on a given computer. This even applies to sound media. The low-order 6 bits specify pixel depths and the upper 2 bits specify quality levels. If a bit is set to 1, the media can be played at the corresponding depth and quality level.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
vrmovies

vrmovies.win

Declared In
`Movies.h`

GetMediaSample

Returns a sample from a movie data file.

```
OSErr GetMediaSample (
   Media theMedia,
   Handle dataOut,
   long maxSizeToGrow,
   long *size,
   TimeValue time,
   TimeValue *sampleTime,
   TimeValue *durationPerSample,
   SampleDescriptionHandle sampleDescriptionH,
   long *sampleDescriptionIndex,
   long maxNumberOfSample,
   long *numberOfSamples,
   short *sampleFlags
);
```

Parameters

theMedia
The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (page 1630) and `GetTrackMedia` (page 1612). See `Media Identifiers`.

dataOut
A handle. The `GetMediaSample` function returns the sample data into this handle. The function increases the size of this handle, if necessary. You can specify the handle's maximum size with the `maxSizeToGrow` parameter.

maxSizeToGrow
The maximum number of bytes of sample data to be returned. The `GetMediaSample` function does not increase the handle specified by the `dataOut` parameter to a size greater than you specify with this parameter. Set this value to 0 to enforce no limit on the number of bytes to be returned.

size
A pointer to a long integer. The `GetMediaSample` function updates the field referred to by the `size` parameter with the number of bytes of sample data returned in the handle specified by the `dataOut` parameter. Set this parameter to `NIL` if you are not interested in this information.

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

sampleTime

A pointer to a time value. The `GetMediaSample` 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 `NIL`.

durationPerSample

A pointer to a time value. The Movie Toolbox returns the duration of each sample in the media. This time value is expressed in the media's time scale. Set this parameter to 0 if you don't want this information.

sampleDescriptionH

A handle to a `SampleDescription` structure. The `GetMediaSample` 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 `GetMediaSample` function returns an index value to the `SampleDescription` structure that corresponds to the returned sample data. You can retrieve the structure by calling `GetMediaSampleDescription` (page 1587) 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 `GetMediaSample` 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 `NIL`.

sampleFlags

A pointer to a short integer in which `GetMediaSample` returns flags (see below) that describe the sample. Unused flags are set to 0. If you don't want this information, set this parameter to `NIL`. See these constants:

`mediaSampleNotSync`

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qdmediahandler.win

qttext.win

qtwiredactions

qtwiredactions.win

vrmakepano

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` (page 1630) and `GetTrackMedia` (page 1612).

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` (page 1587) 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. See these constants:

`mediaSampleNotSync`

Return Value

You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). 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` (page 1583) 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:

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

MovieVideoChart

Declared In

`Movies.h`

GetMediaSampleCount

Determines the number of samples in a media.

```
long GetMediaSampleCount (
   Media theMedia
);
```

Parameters

theMedia

The media for this operation. You obtain this media identifier from such functions as `NewTrackMedia` (page 1630) and `GetTrackMedia` (page 1612).

Return Value

The number of samples in the media.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

MovieVideoChart

vrmakepano

vrmakepano.win

Declared In

`Movies.h`

GetMediaSampleDescription

Retrieves a SampleDescription structure from a media.

```
void GetMediaSampleDescription (
   Media theMedia,
   long index,
   SampleDescriptionHandle descH
);
```

Parameters

theMedia

The media for this operation. You obtain this media identifier from such functions as `NewTrackMedia` (page 1630) and `GetTrackMedia` (page 1612).

index

The index of the `SampleDescription` structure to retrieve. This index corresponds to the structure itself, not to the samples in the media. Index numbers start with 1.

descH

Specifies a handle that is to receive the `SampleDescription` structure. The Movie Toolbox correctly resizes this handle for the returned structure. If there is no description for the specified index, the function returns this handle unchanged. Your application must allocate and dispose of this handle.

Discussion

The Movie Toolbox identifies a media's sample descriptions with an index value, ranging from 1 to the number of sample descriptions in the media. Sample description indexes provide a convenient way to access each sample description in a media. You can access error returns from this function through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). See `Error Codes`.

Special Considerations

The format of sample descriptions differs by media type. Sample descriptions for image data are defined by `ImageDescription` structures. Sample descriptions for sound are defined by `SoundDescription` structures. Sample descriptions for text are defined by `TextDescription` structures.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CompressMovies

DigitizerShell

DragAndDrop Shell

MovieGWorlds

QT Internals

Declared In

`Movies.h`

GetMediaSampleDescriptionCount

Returns the number of sample descriptions in a media.

```
long GetMediaSampleDescriptionCount (
   Media theMedia
);
```

Parameters

theMedia

The media for this operation. You obtain this media identifier from such functions as `NewTrackMedia` (page 1630) and `GetTrackMedia` (page 1612).

Return Value

The number of sample descriptions in the media.

Special Considerations

The format of sample descriptions differs by media type. Sample descriptions for image data are defined by `ImageDescription` structures. Sample descriptions for sound are defined by `SoundDescription` structures. Sample descriptions for text are defined by `TextDescription` structures.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

GetMediaSampleReference

Obtains reference information about samples that are stored in a movie data file.

```
OSErr GetMediaSampleReference (
   Media theMedia,
   long *dataOffset,
   long *size,
   TimeValue time,
   TimeValue *sampleTime,
   TimeValue *durationPerSample,
   SampleDescriptionHandle sampleDescriptionH,
   long *sampleDescriptionIndex,
   long maxNumberOfSamples,
   long *numberOfSamples,
   short *sampleFlags
);
```

Parameters

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (page 1630) and `GetTrackMedia` (page 1612). See `Media Identifiers`.

dataOffset

A pointer to a long integer. `GetMediaSampleReference` updates the field referred to by this parameter with the offset to the sample data. This parameter is used differently by each media handler. For example, the hierarchical file system (HFS) media handler returns an offset into the file that contains the media data.

size

A pointer to a long integer. `GetMediaSampleReference` updates the field referred to by the `size` parameter with the number of bytes of sample data referred to by the reference. Set this parameter to `NIL` if you are not interested in this information.

time

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

sampleTime

A pointer to a time value. `GetMediaSampleReference` 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 `NIL`.

durationPerSample

A pointer to a time value. The Movie Toolbox returns the duration of each sample in the media. This time value is expressed in the media's time scale. Set this parameter to 0 if you don't want this information.

sampleDescriptionH

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

sampleDescriptionIndex

A pointer to a long integer. `GetMediaSampleReference` returns an index value to the `SampleDescription` structure that corresponds to the returned sample data. To retrieve the media sample description, pass this index in the `descH` parameter of `GetMediaSampleDescription` (page 1587). 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 a reference that refers to 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. `GetMediaSampleReference` updates the field referred to by this parameter with the number of samples referred to by the returned reference. If you don't want this information, set this parameter to `NIL`.

sampleFlags

A pointer to a short integer in which `GetMediaSampleReference` returns flags (see below) that describe the samples referred to by the returned reference. Unused flags are set to 0. If you don't want this information, set this parameter to `NIL`. See these constants:

`mediaSampleNotSync`

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

SlideShowImporter

SlideShowImporter.win

Declared In

`Movies.h`

GetMediaSampleReferences

Obtains reference information about groups of samples that are stored in a movie.

```
OSErr GetMediaSampleReferences (
   Media theMedia,
   TimeValue time,
   TimeValue *sampleTime,
   SampleDescriptionHandle sampleDescriptionH,
   long *sampleDescriptionIndex,
   long maxNumberOfEntries,
   long *actualNumberofEntries,
   SampleReferencePtr sampleRefs
);
```

Parameters

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (page 1630) and `GetTrackMedia` (page 1612). See `Media Identifiers`.

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

sampleTime
: A pointer to a time value. `GetMediaSampleReferences` updates this time value to indicate the actual time of the first returned sample data. If you are not interested in this information, set this parameter to `NIL`.

sampleDescriptionH
: A handle to a `SampleDescription` structure. `GetMediaSampleReference` (page 1589) returns the structure corresponding to the returned sample data. The function resizes this handle as appropriate. `GetMediaSampleReferences` only returns a single sample description. If the sample description changes within the media, `GetMediaSampleReferences` returns only as many samples as use a single sample description. You must call it again to get the next group of samples using the next sample description. If you don't want the `SampleDescription` structure, set this parameter to `NIL`.

sampleDescriptionIndex
: A pointer to a long integer. `GetMediaSampleReferences` returns an index value to the `SampleDescription` structures that correspond to the returned sample data. Use this index to retrieve the media sample descriptions with `GetMediaSampleDescription` (page 1587). If you don't want this information, set this parameter to `NIL`.

maxNumberOfEntries
: The maximum number of entries to be returned. The sample references pointer provided by the `sampleRefs` parameter must be large enough to receive the number of entries specified by this parameter. The toolbox does not return more entries than you specify with this parameter. It may, however, return fewer.

actualNumberofEntries
: A pointer to a long integer. `GetMediaSampleReferences` updates the field referred to by this parameter with the number of entries referred to by the returned reference.

sampleRefs
: A pointer to the number of `SampleReferenceRecord` structures specified in the `maxNumberOfEntries` parameter. On return from this call, the number of sample reference records indicated by the value returned in `actualNumberofEntries` will be filled in.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
Using this function instead of `GetMediaSampleReference` (page 1589) can greatly increase the performance of operations that need access to information about each sample in a movie. No information is returned from this call that wasn't previously available from `GetMediaSampleReference`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

GetMediaSampleReferences64

Provides a 64-bit version of GetMediaSampleReferences.

```
OSErr GetMediaSampleReferences64 (
   Media theMedia,
   TimeValue time,
   TimeValue *sampleTime,
   SampleDescriptionHandle sampleDescriptionH,
   long *sampleDescriptionIndex,
   long maxNumberOfEntries,
   long *actualNumberofEntries,
   SampleReference64Ptr sampleRefs
);
```

Parameters

theMedia
: The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (page 1630) and `GetTrackMedia` (page 1612). See `Media Identifiers`.

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

sampleTime
: A pointer to a time value. GetMediaSampleReferences64 updates this time value to indicate the actual time of the first returned sample data. If you are not interested in this information, set this parameter to `NIL`.

sampleDescriptionH
: A handle to a `SampleDescription` structure. `GetMediaSampleReference` (page 1589) returns the sample description corresponding to the returned sample data. The function resizes this handle as appropriate. `GetMediaSampleReferences` only returns a single structure. If the sample description changes within the media, `GetMediaSampleReferences` returns only as many samples as use a single sample description. You must call it again to get the next group of samples using the next sample description. If you don't want the `SampleDescription` structure, set this parameter to `NIL`.

sampleDescriptionIndex
: A pointer to a long integer. GetMediaSampleReferences64 returns an index value to the sample descriptions that correspond to the returned sample data. Use this index to retrieve the media sample descriptions with `GetMediaSampleDescription` (page 1587). If you don't want this information, set this parameter to `NIL`.

maxNumberOfEntries
: The maximum number of entries to be returned. The sample references pointer provided by the `sampleRefs` parameter must be large enough to receive the number of entries specified by this parameter. The toolbox does not return more entries than you specify with this parameter. It may, however, return fewer.

actualNumberofEntries
: A pointer to a long integer. GetMediaSampleReferences64 updates the field referred to by this parameter with the number of entries referred to by the returned reference.

sampleRefs
: A pointer to the number of `SampleReference64Record` structures specified in the `maxNumberOfEntries` parameter. On return from this call, the number of sample reference records indicated by the value returned in `actualNumberofEntries` will be filled in.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
The only difference between this function and `GetMediaSampleReferences` (page 1590) is that the `sampleRefs` parameter points to SampleReference64Record structures instead of `SampleReferenceRecord` structures.

Special Considerations

New applications should use this function instead of the 32-bit version.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

GetMediaShadowSync

Obsolete; no longer supported.

```
OSErr GetMediaShadowSync (
   Media theMedia,
   long frameDiffSampleNum,
   long *syncSampleNum
);
```

Parameters

theMedia
Indicates the media in which the shadow sync sample has been established and from which the shadow sync number is to be obtained.

frameDiffSampleNum
The frame difference sample number associated with the desired shadow sync sample number.

syncSampleNum
A pointer to the sample number of the shadow sync sample. If the media does not have a shadow sync sample, 0 is returned in the `syncSampleNum` parameter.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

GetMediaSyncSampleCount

Gets the number of sync samples in a media.

```
long GetMediaSyncSampleCount (
   Media theMedia
);
```

Parameters

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (page 1630) and `GetTrackMedia` (page 1612). See `Media Identifiers`.

Return Value

The number of sync samples in the media.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

GetMediaTimeScale

Determines a media's time scale.

```
TimeScale GetMediaTimeScale (
   Media theMedia
);
```

Parameters

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (page 1630) and `GetTrackMedia` (page 1612). See `Media Identifiers`.

Return Value

The media's time scale.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

AddFrameToMovie

BurntTextSampleCode

MovieVideoChart

qttext.win

vrmakeobject

Declared In
`Movies.h`

GetMediaTrack

Determines the track that uses a specified media.

```
Track GetMediaTrack (
   Media theMedia
);
```

Parameters

`theMedia`
: The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (page 1630) and `GetTrackMedia` (page 1612). See `Media Identifiers`.

Return Value
The track identifier of the track that uses the specified media.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CaptureAndCompressIPBMovie
QTExtractAndConvertToMovieFile
qtmovietrack
qtmovietrack.win
SCAudioCompress

Declared In
`Movies.h`

GetMediaUserData

Obtains access to a media's user data list.

```
UserData GetMediaUserData (
   Media theMedia
);
```

Parameters

`theMedia`
: The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (page 1630) and `GetTrackMedia` (page 1612). See `Media Identifiers`.

Return Value
The media's user data list.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

GetMovieDataSize

Determines the size of the sample data in a segment of a movie.

```
long GetMovieDataSize (
   Movie theMovie,
   TimeValue startTime,
   TimeValue duration
);
```

Parameters

theMovie
The movie for this operation. You obtain this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

startTime
A time value specifying the starting point of the segment.

duration
A time value that specifies the duration of the segment.

Return Value
The size, in bytes, of the sample data in the defined segment of the designated movie.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

GetMovieDataSize64

Provides a 64-bit version of GetMovieDataSize.

```
OSErr GetMovieDataSize64 (
   Movie theMovie,
   TimeValue startTime,
   TimeValue duration,
   wide *dataSize
);
```

Parameters

theMovie
The movie for this operation. You obtain this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

startTime
A time value specifying the starting point of the segment.

duration
A time value that specifies the duration of the segment.

data size
The size, in bytes, of the sample data in the defined segment of the designated movie.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
The only difference between this function and `GetMovieDataSize` (page 1596) is that the `dataSize` parameter is a 64-bit integer instead of a 32-bit integer.

Special Considerations

New applications should use this function instead of the 32-bit version.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

GetMovieImporterForDataRef

Gets the movie importer component for a movie.

```
OSErr GetMovieImporterForDataRef (
   OSType dataRefType,
   Handle dataRef,
   long flags,
   Component *importer
);
```

Parameters

dataRefType
The type of data reference; see `Data References`.

dataRef
A handle to the `data` reference. The type of information stored in the handle depends upon the `data` reference type specified by `dataRefType`.

flags
Flags (see below) that modify this function's behavior. See these constants:
`kGetMovieImporterDontConsiderGraphicsImporters`

importer
A pointer to an importer component that can import the movie. Returns `NIL` if no importer can be found.

Return Value

If this function is allowed to use `async` calls (by being passed `kGetMovieImporterUseAsyncCalls` in the `flags` parameter), it returns `notEnoughDataErr` if it would block. You can access this error return through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. For other errors, see `Error Codes`.

Discussion

You can use `GetMovieImporterForDataRef` to determine if a file can be opened by QuickTime as a movie (for example, in a drag-and-drop operation) as illustrated below:

```
AliasHandle             alias;
MovieImportComponent    mi;
NewAliasMinimal(&reply.sfFile, &alias);
GetMovieImporterForDataRef(rAliasType, (Handle)alias,
kGetMovieImporterDontConsiderGraphicsImporters, &mi);
DisposeHandle((Handle)alias);
if (mi !=NIL) {
    // this file can be opened as a movie
    . . .
    }
```

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

MakeEffectMovie

qteffects.win

qtgraphics.win

qtwiredactions

vrbackbuffer.win

Declared In

`Movies.h`

GetMovieIndTrack

Determines the track identifier of a track, given the track's index value.

```
Track GetMovieIndTrack (
   Movie theMovie,
   long index
);
```

Parameters

`theMovie`

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

`index`

The index value of the track for this operation.

Return Value
A track identifier. If the function cannot locate the track, it sets this returned value to `NIL`.

Discussion
This function returns the track identifier that is appropriate to the specified track. The index value identifies the track among all current tracks in a movie. Index values range from 1 to the number of tracks in the movie. The following code sample illustrates its use:

```
// GetMovieIndTrack coding example
// See "Discovering QuickTime," page 363
Movie           movie1;
TimeValue       lOldDuration;
Movie           movie2;
long            lIndex, lOrigTrackCount, lReferenceIndex;
Track           track, trackSprite;
// get the first track in original movie and position at the start
trackSprite =GetMovieIndTrack(movie1, 1);
SetMovieSelection(movie1, 0, 0);
// remove all tracks except video in modifier movie
for (lIndex =1; lIndex <=GetMovieTrackCount(movie2); lIndex++) {
    Track       track =GetMovieIndTrack(movie2, lIndex);
    OSType      dwType;
    GetMediaHandlerDescription(GetTrackMedia(track),
                                &dwType, NIL, NIL);
    if (dwType !=VideoMediaType) {
        DisposeMovieTrack(track);
        lIndex--;
    }
}
// add the modifier track to original movie
lOldDuration =GetMovieDuration(movie1);
AddMovieSelection(movie1, movie2);
DisposeMovie(movie2);
// truncate the movie to the length of the original track
DeleteMovieSegment(movie1, lOldDuration,
                    GetMovieDuration(movie1) - lOldDuration);
// associate the modifier track with the original sprite track
track =GetMovieIndTrack(movie1, lOrigTrackCount + 1);
AddTrackReference(trackSprite, track, kTrackModifierReference,
                    &lReferenceIndex);
```

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
ChromaKeyMovie

QT Internals

qtinfo

qtinfo.win

qttext.win

Declared In
`Movies.h`

GetMovieIndTrackType

Searches for all of a movie's tracks that share a given media type or media characteristic.

```
Track GetMovieIndTrackType (
   Movie theMovie,
   long index,
   OSType trackType,
   long flags
);
```

Parameters

theMovie

The movie for this operation. Your application obtains this identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

index

The index value of the track for this operation. This is not that same as the track's index value in the movie. Rather, this parameter is an index into the set of tracks that meet your other selection criteria.

trackType

Contains either a media type or a media characteristic value. The toolbox applies this value to the search, and returns information about tracks that meet this criterion. You indicate whether you have specified a media type or characteristic value by setting the `flags` parameter appropriately.

flags

Contains flags (see below) that control the search operation. Note that you may not set both `movieTrackMediaType` and `movieTrackCharacteristic` to 1. See these constants:

`movieTrackMediaType`

`movieTrackCharacteristic`

`movieTrackEnabledOnly`

Return Value

A track identifier.

Discussion

The toolbox returns the track identifier that corresponds to the track that meets your selection criteria. If the toolbox cannot find a matching track, in returns a value of `NIL`. Note that the `index` parameter does not work the same way that is does in `GetMovieIndTrack` (page 1598). With `GetMovieIndTrackType`, the `index` parameter specifies an index into the set of tracks that meet your other selection criteria. For example, in order to find the third track that supports the sound characteristic, you would call the function in the following manner:

```
theTrack =GetMovieIndTrackType (theMovie, 3, AudioMediaCharacteristic,
movieTrackCharacteristic);
```

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

MakeEffectMovie

qteffects.win

qttext

qttext.win
qttimecode.win

Declared In
`Movies.h`

GetMovieTrack

Determines the track identifier of a track, given the track's ID value.

```
Track GetMovieTrack (
   Movie theMovie,
   long trackID
);
```

Parameters

theMovie
: The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

trackID
: The ID value of the track for this operation.

Return Value
A track identifier.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

GetMovieTrackCount

Returns the number of tracks in a movie.

```
long GetMovieTrackCount (
   Movie theMovie
);
```

Parameters

theMovie
: The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

Return Value
The number of tracks in the movie.

Discussion
The following code sample illustrates the use of `GetMovieTrackCount`:

```
// GetMovieTrackCount coding example
// See "Discovering QuickTime," page 363
Movie           movie1;
TimeValue       lOldDuration;
Movie           movie2;
long            lIndex, lOrigTrackCount, lReferenceIndex;
Track           track, trackSprite;
// get the first track in original movie and position at the start
trackSprite =GetMovieIndTrack(movie1, 1);
SetMovieSelection(movie1, 0, 0);
// remove all tracks except video in modifier movie
for (lIndex =1; lIndex <=GetMovieTrackCount(movie2); lIndex++) {
    Track       track =GetMovieIndTrack(movie2, lIndex);
    OSType      dwType;
    GetMediaHandlerDescription(GetTrackMedia(track),
                                &dwType, NIL, NIL);
    if (dwType !=VideoMediaType) {
        DisposeMovieTrack(track);
        lIndex--;
    }
}
// add the modifier track to original movie
lOldDuration =GetMovieDuration(movie1);
AddMovieSelection(movie1, movie2);
DisposeMovie(movie2);
// truncate the movie to the length of the original track
DeleteMovieSegment(movie1, lOldDuration,
                    GetMovieDuration(movie1) - lOldDuration);
// associate the modifier track with the original sprite track
track =GetMovieIndTrack(movie1, lOrigTrackCount + 1);
AddTrackReference(trackSprite, track, kTrackModifierReference,
                    &lReferenceIndex);
```

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QT Internals

qtinfo

qtinfo.win

qttext

qttext.win

Declared In
`Movies.h`

GetNextTrackReferenceType

Determines all of the track reference types that are defined for a given track.

```
OSType GetNextTrackReferenceType (
   Track theTrack,
   OSType refType
);
```

Parameters

theTrack
: Identifies the track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

refType
: The type of reference. Set this parameter to 0 to retrieve the first track reference type. On subsequent requests, use the previous value returned by this function.

Return Value
An `OSType` containing the next track reference type value defined for the track; see `Data References`.

Discussion
There is no implied ordering of the values returned by this function . When you reach the end of the track's reference types, this function sets the returned value to 0.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
MakeEffectMovie

qteffects.win

qtgraphics.win

QTKitTimeCode

vrbackbuffer.win

Declared In
`Movies.h`

GetTrackAlternate

Determines all the tracks in an alternate group.

```
Track GetTrackAlternate (
   Track theTrack
);
```

Parameters

theTrack
: The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

Return Value
The track identifier of the next track in the group.

Discussion

This function returns the track identifier of the next track in the group. Because the alternate group list is circular, you must specify a different track in the group each time you call this function. You have retrieved all the tracks in the group when the function returns the track identifier that you supplied the first time you called `GetTrackAlternate`. If there is only one track in an alternate group, or if the track you specify does not belong to a group, this function returns the track identifier you supply.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

GetTrackCreationTime

Returns a track's creation date and time.

```
unsigned long GetTrackCreationTime (
   Track theTrack
);
```

Parameters

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

Return Value

The track's creation date and time.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

GetTrackDataSize

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

```
long GetTrackDataSize (
   Track theTrack,
   TimeValue startTime,
   TimeValue duration
);
```

Parameters

theTrack
: The track for this operation. You obtain this track identifier from such functions as NewMovieTrack (page 1628) and GetMovieTrack (page 1601).

startTime
: A time value specifying the starting point of the segment.

duration
: A time value that specifies the duration of the segment.

Return Value
The size, in bytes, of the sample data in a segment of a track.

Discussion
This function counts each use of a sample. That is, if a track uses a given sample more than once, the size of that sample is included in the returned size value one time for each use. Consequently, the returned size is greater than or equal to the actual size of the track's sample data.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

GetTrackDataSize64

Provides a 64-bit version of GetTrackDataSize.

```
OSErr GetTrackDataSize64 (
   Track theTrack,
   TimeValue startTime,
   TimeValue duration,
   wide *dataSize
);
```

Parameters

theTrack
: A track identifier. Your application obtains this identifier from such functions as NewMovieTrack (page 1628) and GetMovieTrack (page 1601).

startTime
: A time value specifying the starting point of the segment.

duration
: A time value that specifies the duration of the segment.

dataSize

The size, in bytes, of the sample data in a segment of a track. This function counts each use of a sample. That is, if a track uses a given sample more than once, the size of that sample is included in the returned size value one time for each use. Consequently, the returned size is greater than or equal to the actual size of the track's sample data.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

The only difference between this function and `GetTrackDataSize` (page 1604) is that size of the sample data is returned as a 64-bit integer in the `dataSize` parameter instead of as a 32-bit integer returned by the function.

Special Considerations

New applications should use this function instead of the 32-bit version.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

GetTrackDimensions

Determines a track's source rectangle.

```
void GetTrackDimensions (
   Track theTrack,
   Fixed *width,
   Fixed *height
);
```

Parameters

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` and `GetMovieTrack` (page 1601).

width

A pointer to a fixed-point number. The Movie Toolbox returns the width, in pixels, of the track's rectangle. This value corresponds to the x coordinate of the lower-right corner of the track's rectangle.

height

A pointer to a fixed-point number. The Movie Toolbox returns the height, in pixels, of the track's rectangle. This value corresponds to the y coordinate of the lower-right corner of the track's rectangle.

Return Value

You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qteffects.win

qttext.win

SlideShowImporter.win

vrmakeobject

vrmakepano

Declared In
`Movies.h`

GetTrackDisplayMatrix

Returns a matrix that is the concatenation of all matrices currently affecting the track's location, scaling, and so on, including the movie's matrix, the track's matrix, and the modifier matrix.

```
OSErr GetTrackDisplayMatrix (
   Track theTrack,
   MatrixRecord *matrix
);
```

Parameters

theTrack
The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

matrix
A pointer to a matrix structure.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
Since modifier information is passed between tracks at `MoviesTask` (page 257) time, the information returned by this call represents the matrix in effect at the last `MoviesTask` call.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

GetTrackDuration

Returns the duration of a track.

```
TimeValue GetTrackDuration (
   Track theTrack
);
```

Parameters

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

Return Value

The duration of the specified track, expressed in the time scale of the movie that contains the track.

Discussion

The duration corresponds to the ending time of the track in the movie's time coordinate system (remember that all tracks start at movie time 0).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

BurntTextSampleCode

MakeEffectMovie

makeeffectslideshow

makeeffectslideshow.win

qteffects.win

Declared In

`Movies.h`

GetTrackEditRate

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

```
Fixed GetTrackEditRate (
   Track theTrack,
   TimeValue atTime
);
```

Parameters

theTrack

The track identifier for which the rate of a track edit (at the time given in the `atTime` parameter) is to be determined.

atTime

Indicates a time value 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 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
addflashactions.win

addhtactions.win

addvractions

qtwiredactions

qtwiredactions.win

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` (page 1628) and `GetMovieTrack` (page 1601).

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.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`Movies.h`

GetTrackEnabled

Determines whether a track is currently enabled.

```
Boolean GetTrackEnabled (
   Track theTrack
);
```

Parameters

theTrack
The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` and `GetMovieTrack` (page 1601).

Return Value
TRUE if the specified track is currently enabled, FALSE otherwise.

Discussion
The Movie Toolbox services only enabled tracks.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qttext
qttext.win
qttimecode
qttimecode.win
vrscript.win

Declared In
`Movies.h`

GetTrackID

Determines a track's unique track ID value.

```
long GetTrackID (
   Track theTrack
);
```

Parameters

theTrack
The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

Return Value
The specified track's unique track ID value.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QT Internals

vrmakepano

VRMakePano Library

vrmakepano.win

Declared In
`Movies.h`

GetTrackLayer

Retrieves a track's layer.

```
short GetTrackLayer (
   Track theTrack
);
```

Parameters

theTrack
: The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

Return Value
The specified track's layer number. Layers with lower numbers appear in front of layers with higher numbers.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
BurntTextSampleCode

qtaddeffectseg

qtaddeffectseg.win

qteffects

qteffects.win

Declared In
`Movies.h`

GetTrackMatrix

Retrieves a track's transformation matrix.

```
void GetTrackMatrix (
   Track theTrack,
   MatrixRecord *matrix
);
```

Parameters

theTrack
: The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` and `GetMovieTrack` (page 1601).

matrix

A pointer to a `MatrixRecord` structure. The `GetTrackMatrix` function returns the track's matrix into the structure referred to by this parameter.

Return Value

You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

BurntTextSampleCode

QTKitTimeCode

qttext.win

qttimecode

qttimecode.win

Declared In

`Movies.h`

GetTrackMedia

Determines the media that contains a track's sample data.

```
Media GetTrackMedia (
   Track theTrack
);
```

Parameters

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

Return Value

The media identifier for the media that contains the track's sample data. If the function could not locate the media, it sets this returned value to `NIL`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

MakeEffectMovie

MovieVideoChart

qtspritesplus.win

qttext

qttext.win

Declared In
`Movies.h`

GetTrackModificationTime

Returns a track's modification date and time.

```
unsigned long GetTrackModificationTime (
   Track theTrack
);
```

Parameters

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

Return Value
The specified track's modification date and time.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

GetTrackMovie

Determines the movie that contains a specified track.

```
Movie GetTrackMovie (
   Track theTrack
);
```

Parameters

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` and `GetMovieTrack` (page 1601).

Return Value
The identifier of the movie that contains the track.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
MakeEffectMovie
qtmovietrack
qtmovietrack.win

qttext

qttext.win

Declared In

`Movies.h`

GetTrackOffset

Determines the time difference between the start of a track and the start of the movie that contains the track.

```
TimeValue GetTrackOffset (
   Track theTrack
);
```

Parameters

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

Return Value

The time difference between the start of the specified track and the start of the movie that contains the track.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

addflashactions.win

addhtactions.win

BurntTextSampleCode

qtwiredactions

qtwiredactions.win

Declared In

`Movies.h`

GetTrackReference

Retrieves the track identifier contained in an existing track reference.

```
Track GetTrackReference (
   Track theTrack,
   OSType refType,
   long index
);
```

Parameters

theTrack

Identifies the track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

refType
: The type of reference; see `Data References`.

index
: The index value of the reference found. You obtain this index value when you create the track reference.

Return Value
The track identifier for the specified track. If the toolbox cannot locate the track reference corresponding to your specifications, it returns a value of `NIL`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
MakeEffectMovie
qteffects.win
QTKitTimeCode
qttext
qttext.win

Declared In
`Movies.h`

GetTrackReferenceCount

Determines how many track references of a given type exist for a track.

```
long GetTrackReferenceCount (
   Track theTrack,
   OSType refType
);
```

Parameters

theTrack
: Identifies the track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

refType
: The type of reference; see `Data References`. The toolbox determines the number of track references of this type.

Return Value
A long integer that specifies the number of track references of the specified type in the track. If there are no references of the type you have specified, the function returns a value of 0.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
MakeEffectMovie

qteffects.win

QTKitTimeCode

qttext

qttext.win

Declared In

`Movies.h`

GetTrackSoundLocalizationSettings

Returns a handle to a copy of the current 3D sound settings for a specified track.

```
OSErr GetTrackSoundLocalizationSettings (
   Track theTrack,
   Handle *settings
);
```

Parameters

theTrack

Identifies the track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

settings

A handle to a copy of the current 3D sound settings for a specified track, in the format of an `SSpLocalizationData` record. If there are no 3D sound settings, the returned handle is set to `NIL`.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Special Considerations

The caller of this function is responsible for disposing of the returned handle.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

GetTrackUsage

Determines whether a track is used in a movie, its preview, its poster, or a combination of these.

```
long GetTrackUsage (
   Track theTrack
);
```

Parameters

theTrack
The track for this operation. Your application obtains this track identifier from such functions as NewMovieTrack (page 1628) and GetMovieTrack (page 1601).

Return Value

Track usage flags (see below). These flags may be combined.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

QT Internals

qtinfo

qtinfo.win

Declared In

Movies.h

GetTrackUserData

Obtains access to a track's user data list.

```
UserData GetTrackUserData (
   Track theTrack
);
```

Parameters

theTrack
The track for this operation. Your application obtains this track identifier from such functions as NewMovieTrack (page 1628) and GetMovieTrack (page 1601).

Return Value

A reference to the specified track's user data. If the function could not locate the track's user data, it sets this returned value to NIL.

Discussion

This function returns a reference to the track's user data list, which is valid until you dispose of the track. When you save the track, the Movie Toolbox saves the user data as well.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

MakeEffectMovie

qteffects.win
qtgraphics.win
qtwiredactions
vrbackbuffer.win

Declared In
`Movies.h`

GetTrackVolume

Returns a track's current volume setting.

```
short GetTrackVolume (
   Track theTrack
);
```

Parameters

theTrack
The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

Return Value
The specified track's current volume setting. The values returned in the high and low words range from 0x0000 (silence) to 0x0100 (full volume). You can use constants (see below) to test for full volume and no volume.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
BurntTextSampleCode
MovieGWorlds
QT Internals
vrmakepano
VRMakePano Library

Declared In
`Movies.h`

InsertEmptyMovieSegment

Adds an empty segment to a movie.

```
OSErr InsertEmptyMovieSegment (
   Movie dstMovie,
   TimeValue dstIn,
   TimeValue dstDuration
);
```

Parameters

dstMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), or `NewMovieFromHandle` (page 1400).

dstIn

A time value that specifies where the segment is to be inserted. This time value must be expressed in the movie's time scale.

dstDuration

A time value that specifies the duration of the segment to be added. This time value must be expressed in the movie's time scale.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

You specify the starting time and duration of the empty segment to be added. These times must be expressed in the movie's time scale. You cannot add empty space to the end of a movie. If you want to insert a segment beyond the end of a movie, use `InsertMovieSegment` (page 1622).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

InsertEmptyTrackSegment

Adds an empty segment to a track.

```
OSErr InsertEmptyTrackSegment (
   Track dstTrack,
   TimeValue dstIn,
   TimeValue dstDuration
);
```

Parameters

dstTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

dstIn

A time value specifying where the segment is to be inserted. This time value must be expressed in the time scale of the movie that contains the destination track.

dstDuration

A time value that specifies the duration of the segment to be added. This time value must be expressed in the time scale of the movie that contains the destination track.

Return Value

See `Error Codes`. If you try to add an empty segment beyond the end of a track, this function does not add the empty segment and returns a result code of `invalidTime`. Returns `noErr` if there is no error.

Discussion

You specify the starting time and duration of the empty segment to be added. These times must be expressed in the movie's time scale. This function then inserts the appropriate amount of empty time into the track. The exact meaning of the term empty time depends upon the type of track. For example, empty time in a sound track is silence. Note that you cannot add empty space to the end of a movie or to the end of a track.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

InsertMediaIntoTrack

Inserts a reference to a media segment into a track.

```
OSErr InsertMediaIntoTrack (
   Track theTrack,
   TimeValue trackStart,
   TimeValue mediaTime,
   TimeValue mediaDuration,
   Fixed mediaRate
);
```

Parameters

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) or `GetMovieTrack` (page 1601).

trackStart

A time value specifying where the segment is to be inserted. This time value must be expressed in the movie's time scale. If you set this parameter to -1, the media data is added to the end of the track.

mediaTime

A time value specifying the starting point of the segment in the media. This time value must be expressed in the media's time scale.

mediaDuration

A time value specifying the duration of the media's segment. This time value must be expressed in the media's time scale.

mediaRate

The media's rate. A value of 1.0 indicates the media's natural playback rate. This value should be positive and not 0.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

You specify the segment in the media by providing a starting time and duration. You specify the point in the destination track by providing a time in the track. `InsertMediaIntoTrack` then inserts the media segment into the track at the specified location. The Movie Toolbox determines the duration of the segment in the track based on the media rate and duration information you provide.

Use this function after you have added samples to a media. If you play the track before you call this function, the track does not contain the new media data.

Here's an example of using this function to add atom containers to a track:

```
//InsertMediaIntoTrack coding example
long descSize;
QTVRSampleDescriptionHandle qtvrSampleDesc;

// Create a QTVR sample description handle
descSize =sizeof(QTVRSampleDescription) + GetHandleSize((Handle) vrWorld)
                            - sizeof(UInt32);
qtvrSampleDesc =(QTVRSampleDescriptionHandle) NewHandleClear (descSize);
(*qtvrSampleDesc)->
size =descSize;
(*qtvrSampleDesc)->
type =kQTVRQTVRType;

// Copy the VR world atom container data into the QTVR sample description
BlockMove (*((Handle) vrWorld), &((*qtvrSampleDesc)->
data),
                            GetHandleSize((Handle) vrWorld));
// Now add it to the QTVR track's media
err =BeginMediaEdits (qtvrMedia);
err =AddMediaSample (qtvrMedia, (Handle) nodeInfo, 0,
    GetHandleSize((Handle) nodeInfo), duration,
    (SampleDescriptionHandle) qtvrSampleDesc, 1, 0, &sampleTime);
err =EndMediaEdits (qtvrMedia);
InsertMediaIntoTrack (qtvrTrack, trackTime, sampleTime, duration, 1L<<16);
```

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qteffects

qteffects.win

vrmakepano

VRMakePano Library

vrmakepano.win

Declared In

`Movies.h`

InsertMovieSegment

Copies part of one movie to another.

```
OSErr InsertMovieSegment (
   Movie srcMovie,
   Movie dstMovie,
   TimeValue srcIn,
   TimeValue srcDuration,
   TimeValue dstIn
);
```

Parameters

srcMovie
: The source movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400). This function obtains the movie segment from the source movie specified in this parameter.

dstMovie
: The destination movie for this operation. The function places a copy of the segment, which it obtained from the source movie, into this destination movie.

srcIn
: The start of the segment in the source movie. This time value must be expressed in the source movie's time scale.

srcDuration
: The duration of the segment in the source movie. This time value must be expressed in the source movie's time scale.

dstIn
: A time value specifying where the segment is to be inserted. This time value must be expressed in the destination movie's time scale.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

If you are not copying data from one location in a movie to a different point in the same movie, this function may create new tracks, as appropriate. Before adding a track to the destination movie, the Movie Toolbox looks in the destination movie for tracks that have the same characteristics as the tracks in the source movie. The toolbox considers several characteristics when searching for an appropriate track, including track spatial dimensions, track matrix, track clipping region, track matte, alternate group affiliation, media time scale, media type, media language, and data reference (that is, referring to the same file). If the Movie Toolbox cannot find an appropriate track in the destination movie, it creates a new track with the proper characteristics.

Special Considerations

If you have assigned a progress function to the destination movie, the Movie Toolbox calls that progress function during long copy operations. Some Movie Toolbox functions can take a long time to execute. For example, if you call `FlattenMovie` (page 1336) and specify a large movie, the Movie Toolbox must read and write all the sample data for the movie. During such operations you may wish to display some kind of progress indicator to the user.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
bMoviePalette
bMoviePaletteCocoa
qtstreamsplicer
qtstreamsplicer.win

Declared In
`Movies.h`

InsertTrackSegment

Copies data into a track.

```
OSErr InsertTrackSegment (
   Track srcTrack,
   Track dstTrack,
   TimeValue srcIn,
   TimeValue srcDuration,
   TimeValue dstIn
);
```

Parameters

`srcTrack`
The source track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

`dstTrack`
The destination track for this operation. This function places a copy of the segment, which is obtained from the source track, into this destination track. The media for the destination track must be opened for writing by calling `BeginMediaEdits` (page 1549) in order for the data to be copied. If the media is not opened for writing, the segment will be copied by reference. At the end of the editing session, your application must call `EndMediaEdits` (page 1567) if it has called `BeginMediaEdits`.

`srcIn`
The start of the segment in the source track. This time value must be expressed in the time scale of the movie that contains the source track.

`srcDuration`
The duration of the segment in the source track. This time value must be expressed in the time scale of the movie that contains the source track.

`dstIn`
A time value specifying where the segment is to be inserted. This time value must be expressed in the time scale of the movie that contains the destination track.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

If you are copying data between tracks, make sure that the two tracks are of the same type. For example, you cannot copy a segment from a sound track into a video track. If you have assigned a progress function to the movie that contains the destination track, the Movie Toolbox calls that progress function during long copy operations.

Special Considerations

If you copy a segment without calling `BeginMediaEdits` on the destination track's media, the data can be copied later by flattening the movie.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qtaddeffectseg

qtaddeffectseg.win

qteffects

qteffects.win

qtstreamsplicer.win

Declared In

`Movies.h`

IsScrapMovie

Checks the system scrap to find out if it can translate any of the data into a movie.

```
Component IsScrapMovie (
   Track targetTrack
);
```

Parameters

`targetTrack`

The location of the potential target movie track for the data on the system scrap.

Return Value

If `IsScrapMovie` finds an appropriate type, it returns a movie import component that can translate the scrap. Otherwise, it returns 0.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

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` (page 1630) and `GetTrackMedia` (page 1612).

Return Value

TRUE if the media is valid and contains at least one sample with a nonzero display offset; FALSE otherwise.

Availability

Available in Mac OS X v10.3 and later.

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` (page 1630) and `GetTrackMedia` (page 1612).

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` (page 221) and `GetMoviesStickyError` (page 222). 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.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
MovieVideoChart

Declared In
`Movies.h`

MediaDisplayTimeToSampleNum

Finds 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` (page 1630) and `GetTrackMedia` (page 1612).

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` (page 221) and `GetMoviesStickyError` (page 222). 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.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
MovieVideoChart

Declared In
`Movies.h`

MediaTimeToSampleNum

Lets you find the sample that contains the data for a specified time.

```
void MediaTimeToSampleNum (
   Media theMedia,
   TimeValue time,
   long *sampleNum,
   TimeValue *sampleTime,
   TimeValue *sampleDuration
);
```

Parameters

theMedia

The media for this operation. You obtain this media identifier from such functions as `NewTrackMedia` (page 1630) and `GetTrackMedia` (page 1612).

time

The 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 Movie Toolbox returns the sample number that identifies the sample that contains data for the time specified by the `time` parameter.

sampleTime

A pointer to a time value. The `MediaTimeToSampleNum` function updates this time value to indicate the starting time of the sample that contains data for the time specified by the `time` parameter. This time value is expressed in the media's time scale. Set this parameter to `NIL` if you don't want this information.

sampleDuration

A pointer to a time value. The Movie Toolbox returns the duration of the sample that contains data for the time specified by the `time` parameter. This time value is expressed in the media's time scale. Set this parameter to `NIL` if you don't want this information.

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qttext
qttext.win

Declared In
`Movies.h`

NewMovieEditState

Creates an edit state.

```
MovieEditState NewMovieEditState (
   Movie theMovie
);
```

Parameters

`theMovie`
: The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

Return Value
A pointer to a `MovieEditStateRecord` structure. The edit state contains all the information describing a movie's content, including the current selection, the movie's tracks, and the media data associated with those tracks.

Special Considerations

You must dispose of a movie's `MovieEditStateRecord` structures, using `DisposeMovieEditState` (page 1564), before you dispose of the movie itself.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

NewMovieTrack

Creates a new movie track, without a media.

```
Track NewMovieTrack (
   Movie theMovie,
   Fixed width,
   Fixed height,
   short trackVolume
);
```

Parameters

`theMovie`
: The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

`width`
: A fixed number denoting the display width of the track, in pixels.

height

A fixed number denoting the display height of the track, in pixels. Together, the `height` and width parameters define the track's display rectangle. The upper-left corner of this rectangle lies at (0,0) in the movie's rectangle. The height and width parameters therefore establish the lower-right corner of the track's display rectangle. If you are creating a track that is not displayed, such as a sound track, set the `height` and width parameters to 0.

trackVolume

The volume setting of the track as a 16-bit, fixed-point number. The high-order 8 bits specify the integer portion; the low-order 8 bits specify the fractional part. Volume values range from -1.0 to 1.0. Negative values play no sound but preserve the absolute value of the volume setting. Set this parameter to `kFullVolume` to play the track at its full, natural volume. Set this parameter to `kNoVolume` to set the volume to 0. See these constants:

Return Value

The identifier of the new track.

Discussion

Immediately after creating a new track, you should call `NewTrackMedia` (page 1630) to create a media for the track; a track without a media is of no use. The following code sample creates a new sprite track and media, then calls `BeginMediaEdits` (page 1549) to prepare to add samples to the media:

```
// NewMovieTrack coding example
// See "Discovering QuickTime," page 349
#define kSpriteMediaTimeScale           600
track =NewMovieTrack(movie, ((long)lTrackWidth << 16),
                            ((long)lTrackHeight << 16), 0);
media =NewTrackMedia(track, SpriteMediaType,
                            kSpriteMediaTimeScale, NIL, 0);
FailOSErr(BeginMediaEdits(media));
```

Special Considerations

When you add a track to a movie, the Movie Toolbox automatically adjusts the display `Rect` structure of the movie. You may want to detect these changes by calling `GetMovieBox` (page 207) so that you can adjust the size of the movie's display window.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qteffects

qteffects.win

vrmakepano

VRMakePano Library

vrmakepano.win

Declared In

`Movies.h`

NewTrackEditState

Creates a new edit state for a given track.

```
TrackEditState NewTrackEditState (
   Track theTrack
);
```

Parameters

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

Return Value

The track's edit state identifier. If the edit state could not be created, the returned identifier is set to `NIL`. You must dispose of a movie's track edit states, using Use `DisposeTrackEditState` (page 1566), before disposing of the track or of the movie that contains the track.

Discussion

Use the returned identifier with other Movie Toolbox edit state functions, such as `UseTrackEditState` (page 1660). The edit state contains all the information describing a track's content, including the identity of the media data associated with the track and all the track's edit lists.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

NewTrackMedia

Creates a media for a new track.

```
Media NewTrackMedia (
   Track theTrack,
   OSType mediaType,
   TimeScale timeScale,
   Handle dataRef,
   OSType dataRefType
);
```

Parameters

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628).

mediaType

The type of media to create; see `Media Identifiers`. The Movie Toolbox uses this value to find the correct media handler for the new media. If the Movie Toolbox cannot locate an appropriate media handler, it returns an error.

timeScale

Defines the media's time coordinate system.

dataRef
The data reference. This parameter contains a handle to the information that identifies the file that contains this media's data. The type of information stored in that handle depends upon the value of the `dataRefType` parameter. If you are creating a new media that refers to existing media data, you can use the `GetMediaDataRef` (page 1342) function to obtain information about the existing data reference. You can then supply information about that reference to this function. Set this parameter to `NIL` to use the file that is associated with the movie or if the movie does not have a movie file. For example, if you have created the movie using `CreateMovieFile` (page 1316) or `NewMovieFromFile` (page 1398), the Movie Toolbox assumes that the movie's data resides in the file specified at that time. If you have created the movie using the `NewMovieFromScrap` (page 1402) or `NewMovie` (page 259) functions, the movie does not have a movie file.

dataRefType
The type of data reference; see `Data References`. If the data reference is an alias, you must set this parameter to `rAliasType`. See *Inside Macintosh: Files* for more information about aliases and the Alias Manager.

Return Value
A media identifier, referring to the actual data samples used by the track. If the function cannot create a new media, it sets the returned value to `NIL`.

Discussion
The following code sample creates a new sprite track and media, then calls `BeginMediaEdits` (page 1549) to prepare to add samples to the media:

```
// NewTrackMedia coding example
// See "Discovering QuickTime," page 349
#define kSpriteMediaTimeScale           600
track =NewMovieTrack(movie, ((long)lTrackWidth << 16),
                            ((long)lTrackHeight << 16), 0);
media =NewTrackMedia(track, SpriteMediaType,
                            kSpriteMediaTimeScale, NIL, 0);
FailOSErr(BeginMediaEdits(media));
```

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qteffects

qteffects.win

vrmakepano

VRMakePano Library

vrmakepano.win

Declared In
`Movies.h`

OpenADataHandler

Opens a data handler component.

```
OSErr OpenADataHandler (
   Handle dataRef,
   OSType dataHandlerSubType,
   Handle anchorDataRef,
   OSType anchorDataRefType,
   TimeBase tb,
   long flags,
   ComponentInstance *dh
);
```

Parameters

dataRef

A handle to a data reference. The type of information stored in the handle depends upon the `data` reference type specified by the `dataHandlerSubType` parameter.

dataHandlerSubType

Identifies both the type of data reference and, by implication, the `component` subtype value assigned to the data handler components that operate on data references of that type.

anchorDataRef

A handle to the anchor data reference.

anchorDataRefType

The type of the anchor data reference.

tb

The time base for the data handler. Your application obtains this time base identifier from `NewTimeBase` (page 261).

flags

Flags (see below) that indicate the way in which you intend to use the data handler component. Not all data handlers necessarily support all services; for example, some data handler components may not support streaming writes. Set the appropriate flags to 1. See these constants:

`kDataHCanRead`

`kDataHCanWrite`

`kDataHCanStreamingWrite`

dh

A pointer to a field to receive the `ComponentInstance` value of the newly-opened data handler component.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 4.1.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

ElectricImageComponent

ElectricImageComponent.win

Declared In

`Movies.h`

PasteHandleIntoMovie

Takes the contents of a specified handle, together with its type, and pastes it into a specified movie.

```
OSErr PasteHandleIntoMovie (
   Handle h,
   OSType handleType,
   Movie theMovie,
   long flags,
   ComponentInstance userComp
);
```

Parameters

h

The handle to be pasted into the movie indicated by the `theMovie` parameter.

handleType

The data type of the handle specified in the `h` parameter. If the handle is set to 0, the function searches the scrap for a field of the type `handleType`. If both the `h` parameter and the `handleType` parameter are `NIL`, the function uses the first available data from the scrap.

theMovie

The destination movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

flags

A flag (see below) that can further refine conditions of the paste operation. See these constants:

`pasteInParallel`

userComp

The component or an instance of the component that is to perform the conversion of the data into a QuickTime movie. If you want a particular movie import component to perform the conversion, you may pass the component or an instance of that component. Otherwise, set this parameter to 0 to allow the Movie Toolbox to determine the appropriate component. If you pass in a component instance, this function uses it. This allows you to communicate directly with the component before using this function to establish any conversion parameters. If you pass in a component ID, an instance is created and closed within this function.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

If you are just pasting in data from the scrap, it is best to allow this function to retrieve the data from the scrap, rather than doing it yourself. In this way, the function is able to obtain supplemental data from the scrap, if necessary (for example, `'styl'` resources for `'TEXT'`). This function can paste into the current selection in two different ways. If the selection is empty (for example, duration =0), it adds the data with the appropriate duration. If the selection is not empty, the data is added and then scaled to fit into the duration of the selection. The current selection is deleted, unless you set the `pasteInParallel` flag.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

PasteMovieSelection

Places the tracks from one movie into another movie.

```
void PasteMovieSelection (
   Movie theMovie,
   Movie src
);
```

Parameters

theMovie

The destination movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

src

The source movie for this operation. `PasteMovieSelection` places the tracks from this movie in the destination movie.

Return Value

You can access error returns from this function through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). See `Error Codes`.

Discussion

Whenever possible, the Movie Toolbox uses existing tracks to store the data to be pasted. Before adding a track to the destination movie, the Toolbox looks in the destination movie for tracks that have the same characteristics as the tracks in the source movie. It considers several characteristics when searching for an appropriate track, including track spatial dimensions, track matrix, track clipping region, track matte, alternate group affiliation, media time scale, media type, media language, and data reference (that is, the two tracks must refer to the same file). If the Movie Toolbox cannot find an appropriate track in the destination movie, it creates a track with the proper characteristics. It removes any empty tracks from the destination movie after the paste operation.

Special Considerations

If you have assigned a progress function to the destination movie, the Movie Toolbox calls that progress function during long paste operations.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

PtInMovie

Determines whether a specified point lies in the region defined by a movie's final display boundary region after it has been clipped by the movie's display clipping region.

```
Boolean PtInMovie (
   Movie theMovie,
   Point pt
);
```

Parameters

theMovie
: The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

pt
: The point to be checked. This point must be expressed in the movie's local display coordinate system.

Return Value
Returns TRUE if the point is in the movie.

Discussion
This function is accurate at the current movie time.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

PtInTrack

Determines whether a specified point lies in the region defined by a track's display boundary region after it has been clipped by the movie's final display clipping region.

```
Boolean PtInTrack (
   Track theTrack,
   Point pt
);
```

Parameters

theTrack
: The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

pt
: The point to be checked. This point must be expressed in the local display coordinate system of the movie that contains the track.

Return Value
Returns TRUE if the point lies in the track's display space.

Discussion
This function is accurate at the current movie time.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

PutMovieIntoTypedHandle

Takes a movie, or a single track from within that movie, and converts it into a handle of a specified type.

```
OSErr PutMovieIntoTypedHandle (
   Movie theMovie,
   Track targetTrack,
   OSType handleType,
   Handle publicMovie,
   TimeValue start,
   TimeValue dur,
   long flags,
   ComponentInstance userComp
);
```

Parameters

theMovie
: The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

targetTrack
: The track to convert.

handleType
: The type of the new data.

publicMovie
: The actual handle in which to place the new data.

start
: The start time of the segment of the movie or track to be converted.

dur
: The duration of the segment of the movie or track to be converted.

flags
: Condition of the conversion. Set this parameter to 0.

userComp
: Indicates a component or component instance of the movie export component you want to perform the conversion. Otherwise, set this parameter to 0 for the Movie Toolbox to choose the appropriate component. If you pass in a component instance, this function will use it. This allows you to communicate directly with the component before using this function to establish any conversion parameters. If you pass in a component ID, an instance is created and closed within this function.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CompressMovies
DigitizerShell
DragAndDrop Shell
MovieGWorlds
soundsnippets

Declared In
`Movies.h`

QTGetMIMETypeInfo

Retrieves information about a particular MIME type.

```
OSErr QTGetMIMETypeInfo (
   const char *mimeStringStart,
   short mimeStringLength,
   OSType infoSelector,
   void *infoDataPtr,
   long *infoDataSize
);
```

Parameters

mimeStringStart
A pointer to the first character of a string holding the MIME type.

mimeStringLength
The number of characters in the MIME type string. Pascal, C, and nondelimited string buffers can be passed equally well.

infoSelector
A constant (see below) that indicates the type of information being requested. See these constants:
`kQTGetMIMETypeInfoIsQuickTimeMovieType`
`kQTGetMIMETypeInfoIsUnhelpfulType`

infoDataPtr
A pointer to a value to be updated. For current selectors this value is Boolean.

infoDataSize
On input, a pointer to the size of the data being expected; on output, a pointer to the size of the data being retrieved. In all current cases these will be the same size.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.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` (page 1630) and `GetTrackMedia` (page 1612).

`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` (page 221) and `GetMoviesStickyError` (page 222). It returns `paramErr` if there is a bad parameter value, or `noErr` if there is no error.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
MovieVideoChart

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 (page 1630) and GetTrackMedia (page 1612).

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 (page 221) and GetMoviesStickyError (page 222). It returns paramErr if there is a bad parameter value, or noErr if there is no error.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

MovieVideoChart

Declared In

Movies.h

SampleNumToMediaTime

Finds the time at which a specified sample plays.

```
void SampleNumToMediaTime (
   Media theMedia,
   long logicalSampleNum,
   TimeValue *sampleTime,
   TimeValue *sampleDuration
);
```

Parameters

theMedia

The media for this operation. You obtain this media identifier from such functions as NewTrackMedia (page 1630) and GetTrackMedia (page 1612).

logicalSampleNum

The sample number.

sampleTime
A pointer to a time value. The function updates this time value to indicate the starting time of the sample specified by the `logicalSampleNum` parameter. This time value is expressed in the media's time scale. Set this parameter to `NIL` if you don't want this information.

sampleDuration
A pointer to a time value. The Movie Toolbox returns the duration of the sample specified by the `logicalSampleNum` parameter. This time value is expressed in the media's time scale. Set this parameter to `NIL` if you don't want this information.

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

ScaleMovieSegment

Changes the duration of a segment of a movie.

```
OSErr ScaleMovieSegment (
   Movie theMovie,
   TimeValue startTime,
   TimeValue oldDuration,
   TimeValue newDuration
);
```

Parameters

theMovie
The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), or `NewMovieFromHandle` (page 1400).

startTime
The start of the segment. The `oldDuration` parameter specifies the segment's duration. This time value must be expressed in the movie's time scale.

oldDuration
The original duration of the segment in the source movie. This time value must be expressed in the movie's time scale.

newDuration
The new duration of the segment. This time value must be expressed in the movie's time scale. The function alters the segment to accommodate the new duration.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
The Movie Toolbox scales the segment to accommodate the new duration.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
SlideShowImporter
SlideShowImporter.win

Declared In
`Movies.h`

ScaleTrackSegment

Changes the duration of a segment of a track.

```
OSErr ScaleTrackSegment (
   Track theTrack,
   TimeValue startTime,
   TimeValue oldDuration,
   TimeValue newDuration
);
```

Parameters

theTrack
The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

startTime
The start of the segment. The `oldDuration` parameter specifies the segment's duration. This time value must be expressed in the time scale of the movie that contains the track.

oldDuration
The duration of the segment. This time value must be expressed in the time scale of the movie that contains the track.

newDuration
The new duration of the segment. This time value must be expressed in the time scale of the movie that contains the track. The function alters the segment to accommodate the new duration.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
This function does not cause the Movie Toolbox to add data to or remove data from the movie.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
makeeffectslideshow

makeeffectslideshow.win

Declared In
`Movies.h`

SelectMovieAlternates

Instructs the Movie Toolbox to select appropriate tracks immediately.

```
void SelectMovieAlternates (
   Movie theMovie
);
```

Parameters

theMovie
A movie identifier. Your application obtains this identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

SetAutoTrackAlternatesEnabled

Enables or disables automatic track selection by the Movie Toolbox.

```
void SetAutoTrackAlternatesEnabled (
   Movie theMovie,
   Boolean enable
);
```

Parameters

theMovie
The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

enable
Controls automatic track selection. Set this parameter to TRUE to enable automatic track selection. Set this parameter to FALSE to disable automatic track selection.

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

SetMediaDataHandler

Assigns a data handler to a media.

```
OSErr SetMediaDataHandler (
   Media theMedia,
   short index,
   DataHandlerComponent dataHandler
);
```

Parameters

theMedia
The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (page 1630) and `GetTrackMedia` (page 1612). See `Media Identifiers`.

index
Identifies the data reference for this data handler. You provide the index value that corresponds to the data reference. You must set this parameter to 1.

dataHandler
The data handler for the media. This identifier is a component instance that specifies a connection to a data handler component, such as that returned by `GetMediaDataHandler` (page 1571). If the data handler you specify cannot work with the data stored in the media, the function does not change the media's data handler.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
Your application should normally not call this function. The Movie Toolbox assigns a data handler to each media when you load a movie.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

SetMediaDefaultDataRefIndex

Specifies which of a media's data references is to be accessed during an editing session.

```
OSErr SetMediaDefaultDataRefIndex (
   Media theMedia,
   short index
);
```

Parameters

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (page 1630) and `GetTrackMedia` (page 1612). See `Media Identifiers`.

index

The data reference to access. Values of the `index` parameter range from 1 to the number of data references in the media. You can determine the number of data references by calling `GetMediaDataRefCount` (page 1344). Once set, the default data reference index persists. Set this parameter to 0 to revert to the media's default data reference.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

This function allows you to specify the index of the data reference to be edited. After calling this function, you can start editing that data reference by calling `BeginMediaEdits` (page 1549).

Version Notes

Before QuickTime 2.0, the Movie Toolbox did not allow the creation of tracks that had data in several files. Therefore, there was no mechanism for controlling which data reference was affected by a media editing session.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

SetMediaHandler

Assigns a specific media handler to a track.

```
OSErr SetMediaHandler (
   Media theMedia,
   MediaHandlerComponent mH
);
```

Parameters

theMedia

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

mH

A reference to a media handler component. You can obtain this reference from `GetMediaHandler` (page 1577).

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
Your application should not need to call this function. The Movie Toolbox assigns a media handler to each track when you load a movie.

Special Considerations

The Movie Toolbox closes the track's previous media handler and then opens the new one. It is your responsibility to ensure that the media handler you specify can handle the data in the track.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

SetMediaInputMap

Replaces the media's existing input map with a given input map.

```
OSErr SetMediaInputMap (
   Media theMedia,
   QTAtomContainer inputMap
);
```

Parameters

theMedia

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (page 1630) and `GetTrackMedia` (page 1612). See `Media Identifiers`.

inputMap

The media input map for this operation. If the input map is set to `NIL`, the media's input map is reset to an empty input map.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion
Use this function to specify the media you want to set so you can modify or empty its input map. It makes a copy of the input map passed to it. The following sample code illustrates how to update an input map, using this function and `GetMediaInputMap` (page 1579):

```
// SetMediaInputMap coding example
QTAtomContainer inputMap;
QTAtom inputAtom;
OSType inputType;
Media aVideoMedia =GetTrackMedia(aVideoTrack);
GetMediaInputMap (aVideoMedia, &inputMap);
QTInsertChild(inputMap, kParentAtomIsContainer, kTrackModifierInput,
        addedIndex, 0,0, nil, &inputAtom);
inputType =kTrackModifierTypeClip;
QTInsertChild (inputMap, inputAtom, kTrackModifierType, 1, 0,
        sizeof(inputType), &inputType, nil);
SetMediaInputMap(aVideoMedia, inputMap);
```

```
QTDisposeAtomContainer(inputMap);
```

Special Considerations

Use `QTNewAtomContainer` (page 1451) to create an empty input map.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qteffects

qteffects.win

qtshoweffect

qtshoweffect.win

qtspritesplus.win

Declared In

`Movies.h`

SetMediaLanguage

Sets a media's localized language or region code.

```
void SetMediaLanguage (
   Media theMedia,
   short language
);
```

Parameters

`theMedia`

The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (page 1630) and `GetTrackMedia` (page 1612). See `Media Identifiers`.

`language`

The media's language or region code.

Return Value

You can access error returns from this function through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). See `Error Codes`.

Discussion

You should call this function only when you are creating a new media.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

SetMediaPreferredChunkSize

Specifies a maximum chunk size for a media.

```
OSErr SetMediaPreferredChunkSize (
   Media theMedia,
   long maxChunkSize
);
```

Parameters

theMedia

The media for this operation. Your application obtains this media identifier from such functions as NewTrackMedia (page 1630) and GetTrackMedia (page 1612). See Media Identifiers.

maxChunkSize

The maximum chunk size, in bytes.

Return Value

You can access Movie Toolbox error returns through GetMoviesError (page 221) and GetMoviesStickyError (page 222), as well as in the function result. See Error Codes.

Discussion

The term "chunk" refers to the collection of sample data that is added to a movie when you call AddMediaSample (page 1536). When QuickTime loads a movie for playback, it loads the data a chunk at a time. Consequently, both the size and number of chunks in a movie can affect playback performance. The toolbox tries to optimize playback performance by consolidating adjacent sample references into a single chunk, up to the limit you prescribe with this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

SetMediaQuality

Sets a media's quality level value.

```
void SetMediaQuality (
   Media theMedia,
   short quality
);
```

Parameters

theMedia

The media for this operation. Your application obtains this media identifier from such functions as NewTrackMedia (page 1630) and GetTrackMedia (page 1612). See Media Identifiers.

quality

The media's quality value. The quality value indicates the pixel depths at which the media can be played. This even applies to sound media. The low-order 6 bits of the quality value correspond to specific pixel depths. If a bit is set to 1, the media can be played at the corresponding depth. More than one of these bits may be set to 1. The Movie Toolbox uses this quality value to determine which track it selects to play on a given Macintosh computer. You should set this value only when you are creating a new media.

Return Value

You can access error returns from this function through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

SetMediaSampleDescription

Changes the contents of a particular SampleDescription structure of a specified media.

```
OSErr SetMediaSampleDescription (
   Media theMedia,
   long index,
   SampleDescriptionHandle descH
);
```

Parameters

theMedia

The media for this operation. You obtain this media identifier from such functions as `NewTrackMedia` (page 1630) and `GetTrackMedia` (page 1612).

index

The index of the `SampleDescription` structure to be changed. This index corresponds to the `SampleDescription` structure itself, not the samples in the media. This long integer must be between 1 and the largest `SampleDescription` index.

descH

The handle to the `SampleDescription` structure. If there is no description for the specified index, the function returns this handle unchanged.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code
BurntTextSampleCode

Declared In
`Movies.h`

SetMediaShadowSync

Obsolete; no longer supported.

```
OSErr SetMediaShadowSync (
   Media theMedia,
   long frameDiffSampleNum,
   long syncSampleNum
);
```

Parameters

`theMedia`
The media in which the shadow sync is to be created.

`frameDiffSampleNum`
Specifies a frame difference sample. The sample number is obtained from `MediaTimeToSampleNum` (page 1627).

`syncSampleNum`
Specifies a shadow sync sample. The sample number is obtained from `MediaTimeToSampleNum` (page 1627).

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

SetMediaTimeScale

Sets a media's time scale.

```
ComponentResult ADD_MEDIA_BASENAME() SetMediaTimeScale
```

Parameters

`theMedia`
The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (page 1630) and `GetTrackMedia` (page 1612). See `Media Identifiers`.

`timeScale`
The media's new time scale.

Return Value

You can access error returns from this function through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

SetTrackAlternate

Adds tracks to, or remove tracks from, alternate groups.

```
void SetTrackAlternate (
   Track theTrack,
   Track alternateT
);
```

Parameters

theTrack

The track and group for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601). `SetTrackAlternate` changes this track's group affiliation based on the value of the `alternateT` parameter.

alternateT

Controls whether the function adds the track to a group or removes it from a group. If this parameter contains a valid track identifier, the Movie Toolbox adds this track to the group that contains the track specified by the parameter `theTrack`. If the track identified by this parameter already belongs to a group, the Movie Toolbox combines the two groups into a single group.

Return Value

You can access error returns from this function through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

SetTrackDimensions

Establishes a track's source rectangle.

```
void SetTrackDimensions (
   Track theTrack,
   Fixed width,
   Fixed height
);
```

Parameters

theTrack
: The track for this operation. Your application obtains this track identifier from such functions as NewMovieTrack (page 1628) and GetMovieTrack (page 1601).

width
: A fixed-point number that specifies the width, in pixels, of the track's rectangle. This value corresponds to the x coordinate of the lower-right corner of the track's rectangle.

height
: A fixed-point number that specifies the height, in pixels, of the track's rectangle. This value corresponds to the y coordinate of the lower-right corner of the track's rectangle.

Return Value

You can access error returns from this function through GetMoviesError (page 221) and GetMoviesStickyError (page 222). See Error Codes.

Discussion

If you change the dimensions of an existing track, the media data is scaled to fit into the new rectangle.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

QTKitTimeCode

qttimecode.win

SlideShowImporter

SlideShowImporter.win

TimeCode Media Handlers

Declared In

Movies.h

SetTrackEnabled

Enables or disables a track.

```
void SetTrackEnabled (
   Track theTrack,
   Boolean isEnabled
);
```

Parameters

theTrack
: The track for this operation. Your application obtains this track identifier from such functions as NewMovieTrack (page 1628) and GetMovieTrack (page 1601).

isEnabled

Enables or disables the `track`. Set this parameter to TRUE to enable the `track`. Set this parameter to FALSE to disable the track.

Return Value

You can access error returns from this function through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). See `Error Codes`.

Discussion

The Movie Toolbox services only enabled tracks.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

QTKitTimeCode

qttimecode

qttimecode.win

vrmakepano

VRMakePano Library

Declared In

`Movies.h`

SetTrackLayer

Sets a track's layer.

```
void SetTrackLayer (
   Track theTrack,
   short layer
);
```

Parameters

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

layer

The track's layer number. Layers are numbered from -32,768 through 32,767; layers with lower numbers appear in front of layers with higher numbers. When you create a new track, the Movie Toolbox sets its track number to 0.

Return Value

You can access error returns from this function through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

BurntTextSampleCode

qtactiontargets

qtactiontargets.win

qtwiredspritesjr

qtwiredspritesjr.win

Declared In

`Movies.h`

SetTrackMatrix

Establishes a track's transformation matrix.

```
void SetTrackMatrix (
   Track theTrack,
   const MatrixRecord *matrix
);
```

Parameters

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

matrix

A pointer to a `MatrixRecord` structure that contains the track's new matrix. If you set this parameter to `NIL`, the Movie Toolbox uses the identity matrix.

Return Value

You can access error returns from this function through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

QTKitTimeCode

qttext.win

qttimecode

qttimecode.win

qtwiredspritesjr

Declared In

`Movies.h`

SetTrackOffset

Modifies the duration of the empty space that lies at the beginning of a track, thus changing the duration of the entire track.

```
void SetTrackOffset (
   Track theTrack,
   TimeValue movieOffsetTime
);
```

Parameters

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

movieOffsetTime

The track's offset from the start of the movie, and must be expressed in the time scale of the movie that contains the track.

Return Value

You can access error returns from this function through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). See `Error Codes`.

Discussion

All of the tracks in a movie use the movie's time coordinate system. That is, the movie's time scale defines the basic time unit for each of the movie's tracks. Each track begins at the beginning of the movie, but the track's data might not begin until some time value other than 0. This intervening time is represented by blank space. In an audio track the blank space translates to silence; in a video track the blank space generates no visual image. Each track has its own duration. This duration need not correspond to the duration of the movie. Movie duration always equals the maximum duration of all the tracks.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

SetTrackReference

Modifies an existing track reference.

```
OSErr SetTrackReference (
   Track theTrack,
   Track refTrack,
   OSType refType,
   long index
);
```

Parameters

theTrack

Identifies the track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

refTrack

The track to be identified in the track reference. The toolbox uses this information to update the existing track reference.

refType

The type of reference.

index

The index value of the reference to be changed. You obtain this index value when you create the track reference.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

You may change the track reference so that it identifies a different track in the movie.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

SetTrackSoundLocalizationSettings

Applies 3D sound effect data to a track.

```
OSErr SetTrackSoundLocalizationSettings (
   Track theTrack,
   Handle settings
);
```

Parameters

theTrack

Identifies the track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

settings

A handle to the settings you want to apply, in the format of an `SSpLocalizationData` record. You can pass a `NIL` handle to indicate that no 3D sound effects should be used for this track. This function makes a copy of the handle passed, so the caller is responsible for disposing of it.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Discussion

This function replaces the 3D sound settings for the specified track with the new `SSpLocalizationData` record contained in the `settings` parameter. The effect of the new 3D sound setting takes place immediately. This call always stores the new record passed, even if the track or the computer is not capable of actually meeting the request. When the movie is saved, the 3D sound settings are saved with it.

The following example code shows how to set the static 3D sound setting for a track using this function:

```
// SetTrackSoundLocalizationSettings coding example
void setTrackSoundLocalization(Track t)
{
    SSpLocalizationData loc;
    Handle h;
    OSErr err;
    loc.cpuLoad =0;
    loc.medium =kSSpMedium_Air;
    loc.humidity =0;
    loc.roomSize =250;
    loc.roomReflectivity =-5;
    loc.reverbAttenuation =-5;
    loc.sourceMode =kSSpSourceMode_Localized;
    loc.referenceDistance =1;
    loc.coneAngleCos =0;
    loc.coneAttenuation =0;
    loc.currentLocation.elevation =0;
    loc.currentLocation.azimuth =0;
    loc.currentLocation.distance =2;
    loc.currentLocation.projectionAngle =0;
    loc.currentLocation.sourceVelocity =0;
    loc.currentLocation.listenerVelocity =0;
    loc.reserved0 =0;
    loc.reserved1 =0;
    loc.reserved2 =0;
    loc.reserved3 =0;
    loc.virtualSourceCount =0;
    err =PtrToHand(&loc, &h, sizeof(loc));
    err =SetTrackSoundLocalizationSettings(t, h);
    DisposeHandle(h);
}
```

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

SetTrackUsage

Specifies whether a track is used in a movie, its preview, its poster, or a combination of these.

```
void SetTrackUsage (
   Track theTrack,
   long usage
);
```

Parameters

theTrack
: The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

usage
Contains flags (see below) that specify how the track is to be used. Be sure to set unused flags to 0. See these constants:
`trackUsageInMovie`
`trackUsageInPreview`
`trackUsageInPoster`

Return Value
You can access error returns from this function through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qtinfo
qtinfo.win

Declared In
`Movies.h`

SetTrackVolume

Sets a track's current volume.

```
void SetTrackVolume (
   Track theTrack,
   short volume
);
```

Parameters

theTrack
The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

volume
The current volume setting of the track represented as a 16-bit, fixed-point number. The high-order 8 bits contain the integer part of the value; the low-order 8 bits contain the fractional part. Volume values range from -1.0 to 1.0. Negative values play no sound but preserve the absolute value of the volume setting. You can use constants (see below) for full volume and no volume. See these constants:

Return Value
You can access error returns from this function through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222). See `Error Codes`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
soundsnippets
soundsnippets.win
vrscript
vrscript.win

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` (page 1628) and `GetMovieTrack` (page 1601).

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.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
MovieVideoChart

Declared In
`Movies.h`

TrackTimeToMediaTime

Converts a track's time value to a time value that is appropriate to the track's media, using the track's edit list.

```
TimeValue TrackTimeToMediaTime (
   TimeValue value,
   Track theTrack
);
```

Parameters

value

The track's time value; must be expressed in the time scale of the movie that contains the track.

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

Return Value

The track's time value, but 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 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

addhtactions.win

BurntTextSampleCode

qttext

qttext.win

qtwiredactions

Declared In

`Movies.h`

UseMovieEditState

Returns a movie to the condition determined by an edit state created previously.

```
OSErr UseMovieEditState (
   Movie theMovie,
   MovieEditState toState
);
```

Parameters

theMovie

The movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie` (page 259), `NewMovieFromFile` (page 1398), and `NewMovieFromHandle` (page 1400).

toState

The edit state for this operation. Your application obtains this edit state identifier when you create the edit state by calling `NewMovieEditState` (page 1628).

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

UseTrackEditState

Returns a track to the condition determined by an edit state created previously.

```
OSErr UseTrackEditState (
   Track theTrack,
   TrackEditState state
);
```

Parameters

theTrack

The track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` (page 1628) and `GetMovieTrack` (page 1601).

state

The edit state for this operation. Your application obtains this edit state identifier when you create the edit state by calling `NewTrackEditState` (page 1630).

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

Callbacks

Data Types

DataHandlerComponent

Represents a type used by the Track and Media API.

```
typedef Component DataHandlerComponent;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

MediaHandlerComponent

Represents a type used by the Track and Media API.

```
typedef Component MediaHandlerComponent;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

MovieEditState

Represents a type used by the Track and Media API.

```
typedef MovieEditStateRecord * MovieEditState;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

MovieEditStateRecord

Undocumented

```
struct MovieEditStateRecord {
     long    data[1];
 };
```

Fields

`data`

Discussion

Undocumented

Declared In

`Movies.h`

SampleReference64Ptr

Represents a type used by the Track and Media API.

```
typedef SampleReference64Record * SampleReference64Ptr;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Movies.h`

SampleReference64Record

Provides a 64-bit version of SampleReferenceRecord.

```
struct SampleReference64Record {
     wide             dataOffset;
     unsigned long    dataSize;
     TimeValue        durationPerSample;
     unsigned long    numberOfSamples;
     short            sampleFlags;
 };
```

Fields

`dataOffset`

Discussion

Specifies the offset into the movie data file. This field specifies the offset into the file of the sample data.

`dataSize`

Discussion

Specifies the total number of bytes of sample data identified by the reference. All samples referenced by a single SampleReference64Record must be the same size.

`durationPerSample`

Discussion

Specifies the duration of each sample in the reference. You must specify this parameter in the media's time scale. All samples referenced by a single SampleReference64Record must be the same duration.

`numberOfSamples`

Discussion
Specifies the number of samples contained in the reference.

`sampleFlag`

Discussion
Contains flags (see below) that control the operation. Set unused flags to 0. See these constants:

    `mediaSampleNotSync`

Related Functions
`AddMediaSampleReferences64` (page 1544)
`GetMediaSampleReferences64` (page 1592)

Declared In
`Movies.h`

SampleReferencePtr

Represents a type used by the Track and Media API.

```
typedef SampleReferenceRecord * SampleReferencePtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

SampleReferenceRecord

Describes a sample or group of similar samples.

```
struct SampleReferenceRecord {
     long          dataOffset;
     long          dataSize;
     TimeValue     durationPerSample;
     long          numberOfSamples;
     short         sampleFlags;
 };
```

Fields
`dataOffset`

Discussion
Specifies the offset into the movie data file. This field specifies the offset into the file of the sample data.

`dataSize`

Discussion
Specifies the total number of bytes of sample data identified by the reference. All samples referenced by a single `SampleReferenceRecord` must be the same size.

`durationPerSample`

Discussion
Specifies the duration of each sample in the reference. You must specify this parameter in the media's time scale. All samples referenced by a single `SampleReferenceRecord` must be the same duration.

`numberOfSamples`

Discussion
Specifies the number of samples contained in the reference.

`sampleFlag`

Discussion
Contains flags (see below) that control the operation. Set unused flags to 0. See these constants:
`mediaSampleNotSync`

Related Functions
`AddMediaSampleReferences` (page 1543)
`GetMediaSampleReferences` (page 1590)

Declared In
`Movies.h`

TrackEditState

Represents a type used by the Track and Media API.

```
typedef TrackEditStateRecord * TrackEditState;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

TrackEditStateRecord

Contains a track edit state.

```
struct TrackEditStateRecord {
     long    data[1];
 };
```

Fields
`data`

Discussion
An array of data that constitutes a track edit state.

Declared In
`Movies.h`

Constants

GetMovieImporter Flags

Constants that represent <codeVoice>GetMovieImporter</codeVoice> flags.

```
enum {
  kGetMovieImporterValidateToFind = 1L << 0,
  kGetMovieImporterAllowNewFile = 1L << 1,
  kGetMovieImporterDontConsiderGraphicsImporters = 1L << 2,
  kGetMovieImporterDontConsiderFileOnlyImporters = 1L << 6,
  kGetMovieImporterAutoImportOnly = 1L << 10 /* reject aggressive movie importers
 which have dontAutoFileMovieImport set*/
};
```

Declared In
`Movies.h`

AddClonedTrackToMovie Values

Constants passed to AddClonedTrackToMovie.

```
enum {
  kQTCloneShareSamples           = 1 << 0,
  kQTCloneDontCopyEdits          = 1 << 1
};
```

Declared In
`Movies.h`

QTGetMIMETypeInfo Values

Constants passed to QTGetMIMETypeInfo.

```
enum {
  kQTGetMIMETypeInfoIsQuickTimeMovieType = 'moov', /* info is a pointer to a
Boolean*/
  kQTGetMIMETypeInfoIsUnhelpfulType = 'dumb' /* info is a pointer to a Boolean*/
};
```

Declared In
`Movies.h`

GetMovieIndTrackType Values

Constants passed to GetMovieIndTrackType.

```
enum {
  movieTrackMediaType           = 1 << 0,
  movieTrackCharacteristic      = 1 << 1,
  movieTrackEnabledOnly         = 1 << 2
};
```

Declared In
`Movies.h`

movieFileSpecValid

Constants grouped with movieFileSpecValid.

```
enum {
  pasteInParallel               = 1 << 0,
  showUserSettingsDialog        = 1 << 1,
  movieToFileOnlyExport         = 1 << 2,
  movieFileSpecValid            = 1 << 3
};
```

Declared In
`Movies.h`

SetTrackUsage Values

Constants passed to SetTrackUsage.

```
enum {
  trackUsageInMovie             = 1 << 1,
  trackUsageInPreview           = 1 << 2,
  trackUsageInPoster            = 1 << 3
};
```

Declared In
`Movies.h`

Media Identifiers

Identify media types in QuickTime.

```
enum {
  VideoMediaType                = 'vide',
  SoundMediaType                = 'soun',
  TextMediaType                 = 'text',
  BaseMediaType                 = 'gnrc',
  MPEGMediaType                 = 'MPEG',
  MusicMediaType                = 'musi',
  TimeCodeMediaType             = 'tmcd',
  SpriteMediaType               = 'sprt',
  FlashMediaType                = 'flsh',
  MovieMediaType                = 'moov',
  TweenMediaType                = 'twen',
  ThreeDeeMediaType             = 'qd3d',
  SkinMediaType                 = 'skin',
  HandleDataHandlerSubType      = 'hndl',
  PointerDataHandlerSubType     = 'ptr ',
  NullDataHandlerSubType        = 'null',
  ResourceDataHandlerSubType    = 'rsrc',
  URLDataHandlerSubType         = 'url ',
  AliasDataHandlerSubType       = 'alis',
  WiredActionHandlerType        = 'wire'
};
```

Constants

`SoundMediaType`

Sound channel.

Available in Mac OS X v10.0 and later.

Declared in `Movies.h`.

`TextMediaType`

Text media.

Available in Mac OS X v10.0 and later.

Declared in `Movies.h`.

Declared In

`Movies.h`

QuickTime Music Architecture Reference

Framework:	Frameworks/QuickTime.framework
Declared in	QuickTimeMusic.h

Overview

The QuickTime Music Architecture (QTMA) allows QuickTime movies, applications, and other software to play individual musical notes, sequences of notes, and a broad range of sounds from a variety of instruments and synthesizers. With QTMA, you can also import Standard MIDI files and convert them into a QuickTime movie for easy playback.

Functions by Task

Allocating and Using Note Channels

`NADisposeNoteChannel` (page 1717) Deprecated in Mac OS X v10.5
  Deletes a specified note channel.

`NAFindNoteChannelTone` (page 1717) Deprecated in Mac OS X v10.5
  Locates the instrument that best fits a requested tone description for a specific channel.

`NAGetController` (page 1718) Deprecated in Mac OS X v10.5
  Retrieves the controller settings for a note channel.

`NAGetIndNoteChannel` (page 1719) Deprecated in Mac OS X v10.5
  Returns the number of note channels handled by the specified note allocator instance.

`NAGetKnob` (page 1719) Deprecated in Mac OS X v10.5
  Obtains the value of a knob for a given note channel.

`NAGetNoteChannelInfo` (page 1721) Deprecated in Mac OS X v10.5
  Returns the index of the music component for the allocated channel and its part number on that music component.

`NAGetNoteRequest` (page 1722) Deprecated in Mac OS X v10.5
  Retrieves the NoteRequest structure that was passed to a note channel.

`NANewNoteChannel` (page 1724) Deprecated in Mac OS X v10.5
  Requests a new note channel with the qualities described in a NoteRequest structure.

`NANewNoteChannelFromAtomicInstrument` (page 1724) Deprecated in Mac OS X v10.5
  Requests a new note channel for an atomic instrument.

NAPlayNote (page 1729) Deprecated in Mac OS X v10.5
    Plays a note with a specified pitch and velocity on the specified note channel.

NAPrerollNoteChannel (page 1730) Deprecated in Mac OS X v10.5
    Attempts to reallocate the note channel if it was invalid previously.

NAResetNoteChannel (page 1731) Deprecated in Mac OS X v10.5
    Turns off all currently active notes on the note channel and resets all controllers to their default values.

NASendMIDI (page 1732) Deprecated in Mac OS X v10.5
    Sends a MIDI music packet to a synthesizer that contains a specific note channel.

NASetAtomicInstrument (page 1733) Deprecated in Mac OS X v10.5
    Initializes a synthesizer part with an atomic instrument.

NASetController (page 1734) Deprecated in Mac OS X v10.5
    Changes the controller setting on a note channel to a specified value.

NASetInstrumentNumber (page 1735) Deprecated in Mac OS X v10.5
    Initializes initializes a synthesizer part with the specified instrument.

NASetInstrumentNumberInterruptSafe (page 1735) Deprecated in Mac OS X v10.5
    Initializes a synthesizer part with the specified instrument during interrupt time.

NASetKnob (page 1736) Deprecated in Mac OS X v10.5
    Sets a note channel knob to a particular value.

NASetNoteChannelBalance (page 1737) Deprecated in Mac OS X v10.5
    Modifies the pan controller setting for a note channel.

NASetNoteChannelSoundLocalization (page 1738) Deprecated in Mac OS X v10.5
    Passes sound localization data to a note channel.

NASetNoteChannelVolume (page 1738) Deprecated in Mac OS X v10.5
    Sets the volume on the specified note channel.

NAUnrollNoteChannel (page 1741) Deprecated in Mac OS X v10.5
    Marks a note channel as available to be stolen.

Calling Generic Music Component Clients

MusicDerivedSetInstrument (page 1679) Deprecated in Mac OS X v10.5
    The complete instrument defined by the Part structure to the synthesizer.

MusicDerivedSetKnob (page 1680) Deprecated in Mac OS X v10.5
    Called when any of the synthesizer's knobs are altered.

MusicDerivedSetMIDI (page 1681) Deprecated in Mac OS X v10.5
    Sets the MIDI channel and other MIDI settings for MIDI output only.

MusicDerivedSetPart (page 1682) Deprecated in Mac OS X v10.5
    Sets the polyphony for the part specified in the GCPart structure.

Managing Instruments and Parts

MusicGetInstrumentAboutInfo (page 1691) Deprecated in Mac OS X v10.5
    Obtains the information about an instrument that appears in its About box.

`MusicGetInstrumentInfo` (page 1692) Deprecated in Mac OS X v10.5
Obtains a list of instruments supported by a synthesizer.

`MusicGetPart` (page 1698) Deprecated in Mac OS X v10.5
Returns the MIDI channel and maximum polyphony for a particular part.

`MusicGetPartAtomicInstrument` (page 1699) Deprecated in Mac OS X v10.5
Returns the atomic instrument currently in a part.

`MusicGetPartController` (page 1700) Deprecated in Mac OS X v10.5
Returns the value of a specified controller on a specified part.

`MusicGetPartInstrumentNumber` (page 1701) Deprecated in Mac OS X v10.5
Returns the instrument number currently assigned to a part.

`MusicGetPartKnob` (page 1701) Deprecated in Mac OS X v10.5
Retrieves the current value of a knob for a part.

`MusicGetPartName` (page 1702) Deprecated in Mac OS X v10.5
Returns the string name of a part.

`MusicResetPart` (page 1703) Deprecated in Mac OS X v10.5
Silences all sounds on a specified part and resets all controllers on that part to their default values.

`MusicSetPart` (page 1707) Deprecated in Mac OS X v10.5
Sets the MIDI channel and maximum polyphony for a specified part.

`MusicSetPartAtomicInstrument` (page 1708) Deprecated in Mac OS X v10.5
Initializes a part with an atomic instrument.

`MusicSetPartController` (page 1709) Deprecated in Mac OS X v10.5
Initializes the value of a specified controller on a specified part.

`MusicSetPartInstrumentNumber` (page 1710) Deprecated in Mac OS X v10.5
Superseded by MusicSetPartInstrumentNumberInterruptSafe.

`MusicSetPartInstrumentNumberInterruptSafe` (page 1710) Deprecated in Mac OS X v10.5
Initializes a part with a particular instrument.

`MusicSetPartKnob` (page 1711) Deprecated in Mac OS X v10.5
Sets a knob for a specified part.

`MusicSetPartName` (page 1711) Deprecated in Mac OS X v10.5
Changes the name of an instrument in a specified part.

`MusicSetPartSoundLocalization` (page 1712) Deprecated in Mac OS X v10.5
Passes sound localization data to a specified synthesizer part.

`MusicStorePartInstrument` (page 1714) Deprecated in Mac OS X v10.5
Puts whatever instrument is on the specified part into the synthesizer's instrument store.

Managing Synthesizers

`MusicFindTone` (page 1684) Deprecated in Mac OS X v10.5
Returns the number of the best-matching instrument provided by a specified music component.

`MusicGetDescription` (page 1688) Deprecated in Mac OS X v10.5
Returns a structure describing the synthesizer controlled by the music component device.

`MusicGetDeviceConnection` (page 1689) Deprecated in Mac OS X v10.5
Determines how many hardware synthesizers are available to a music component and gets the IDs for those devices.

`MusicGetDrumKnobDescription` (page 1689) Deprecated in Mac OS X v10.5
Returns a description of a drum kit knob.

`MusicGetInstrumentKnobDescription` (page 1693) Deprecated in Mac OS X v10.5
Obtains the description of an instrument knob.

`MusicGetKnob` (page 1694) Deprecated in Mac OS X v10.5
Returns the value of the specified global synthesizer knob.

`MusicGetKnobDescription` (page 1695) Deprecated in Mac OS X v10.5
Returns a pointer to an initialized knob description structure describing a global synthesizer knob.

`MusicGetKnobSettingStrings` (page 1696) Deprecated in Mac OS X v10.5
Returns a list of knob setting names known by the specified music component.

`MusicGetMIDIPorts` (page 1697) Deprecated in Mac OS X v10.5
Returns the number of input and output ports a MIDI device has.

`MusicGetMIDIProc` (page 1698) Deprecated in Mac OS X v10.5
Returns a pointer to the procedure a music component is using to process external MIDI notes.

`MusicPlayNote` (page 1702) Deprecated in Mac OS X v10.5
Plays a note on a specified part at a specified pitch and velocity.

`MusicSendMIDI` (page 1704) Deprecated in Mac OS X v10.5
Sends a MIDI packet to a specified port.

`MusicSetKnob` (page 1705) Deprecated in Mac OS X v10.5
Modifies the value of the specified global synthesizer knob.

`MusicSetMIDIProc` (page 1706) Deprecated in Mac OS X v10.5
Informs the music component what procedure to call when it needs to send MIDI data.

`MusicUseDeviceConnection` (page 1715) Deprecated in Mac OS X v10.5
Tells a music component which hardware synthesizer to talk to.

Managing the Generic Music Component

`MusicGenericConfigure` (page 1685) Deprecated in Mac OS X v10.5
Informs the generic music component what services your music component requires and points to any resources that are necessary.

MIDI Component Functions

`QTMIDIGetMIDIPorts` (page 1743) Deprecated in Mac OS X v10.5
Returns two lists of MIDI ports supported by the specified MIDI component: a list of ports that can receive MIDI input and a list of ports that can send MIDI output.

`QTMIDISendMIDI` (page 1744) Deprecated in Mac OS X v10.5
Sends MIDI data to a MIDI port.

`QTMIDIUseSendPort` (page 1745) Deprecated in Mac OS X v10.5
Allocates a MIDI port for output or to release the port.

Miscellaneous Music Component Functions

`MusicGetMasterTune` (page 1696) Deprecated in Mac OS X v10.5
: Returns the synthesizer's master tuning as a fixed-point value in semitones.

`MusicSetMasterTune` (page 1705) Deprecated in Mac OS X v10.5
: Alters a synthesizer's master tuning.

`MusicSetOfflineTimeTo` (page 1707) Deprecated in Mac OS X v10.5
: Advances the synthesizer clock when the synthesizer is not running in real time.

`MusicStartOffline` (page 1713) Deprecated in Mac OS X v10.5
: Informs the QuickTime music synthesizer that the music will not be played through the speakers.

`MusicTask` (page 1714) Deprecated in Mac OS X v10.5
: Allows a music component to perform tasks it must perform at foreground task time.

Note Allocator Configuration and Utilities

`NAGetMIDIPorts` (page 1720) Deprecated in Mac OS X v10.5
: The MIDI input and output ports available to a note allocator.

`NAGetRegisteredMusicDevice` (page 1722) Deprecated in Mac OS X v10.5
: Returns details about music components registered to the specified note allocator instance.

`NARegisterMusicDevice` (page 1730) Deprecated in Mac OS X v10.5
: Registers a music component with the note allocator.

`NASaveMusicConfiguration` (page 1732) Deprecated in Mac OS X v10.5
: Saves the current list of registered devices to a file.

`NATask` (page 1740) Deprecated in Mac OS X v10.5
: Called periodically to allow the note allocator to perform tasks in foreground task time.

`NAUnregisterMusicDevice` (page 1740) Deprecated in Mac OS X v10.5
: Removes a previously registered music component from the note allocator.

Note Allocator Interface Tools

`NACopyrightDialog` (page 1716) Deprecated in Mac OS X v10.5
: Displays a copyright dialog box with information specific to a music device.

`NAPickArrangement` (page 1725) Deprecated in Mac OS X v10.5
: Displays a dialog box to allow instrument selection.

`NAPickEditInstrument` (page 1726) Deprecated in Mac OS X v10.5
: Presents a user interface for changing the instrument in a live note channel or modifying an atomic instrument.

`NAPickInstrument` (page 1728) Deprecated in Mac OS X v10.5
: Presents a user interface for picking an instrument.

`NAStuffToneDescription` (page 1739) Deprecated in Mac OS X v10.5
: Initializes a tone description structure with the details of a General MIDI note channel.

Using the Tune Player

`TuneGetIndexedNoteChannel` (page 1746) Deprecated in Mac OS X v10.5
  Determines how many parts a tune is playing and which instrument is assigned to those parts.

`TuneGetNoteAllocator` (page 1746) Deprecated in Mac OS X v10.5
  Returns the instance of the note allocator that the tune player is using.

`TuneGetPartMix` (page 1747) Deprecated in Mac OS X v10.5
  Gets volume, balance, and mixing settings for a specified part of a tune.

`TuneGetStatus` (page 1748) Deprecated in Mac OS X v10.5
  Returns an initialized structure describing the state of the tune player instance.

`TuneGetTimeBase` (page 1748) Deprecated in Mac OS X v10.5
  Returns the time base of the tune player.

`TuneGetTimeScale` (page 1749) Deprecated in Mac OS X v10.5
  Returns the current time scale for a specified tune player instance.

`TuneGetVolume` (page 1749) Deprecated in Mac OS X v10.5
  Returns the volume associated with an entire tune sequence.

`TuneInstant` (page 1750) Deprecated in Mac OS X v10.5
  Plays a particular sequence of events active at a specified position.

`TunePreroll` (page 1751) Deprecated in Mac OS X v10.5
  Prepares to play a tune player sequence data by attempting to reserve note channels for each part in the sequence.

`TuneQueue` (page 1751) Deprecated in Mac OS X v10.5
  Places a sequence of music events into a queue to be played.

`TuneSetBalance` (page 1752) Deprecated in Mac OS X v10.5
  Modifies the pan controller setting for a tune player.

`TuneSetHeader` (page 1753) Deprecated in Mac OS X v10.5
  Prepares the tune player to accept subsequent music event sequences by defining one or more parts to be used by sequence Note events.

`TuneSetHeaderWithSize` (page 1754) Deprecated in Mac OS X v10.5
  Similar to TuneSetHeader but lets you specify the header length.

`TuneSetNoteChannels` (page 1755) Deprecated in Mac OS X v10.5
  Assigns note channels to a tune player.

`TuneSetPartMix` (page 1755) Deprecated in Mac OS X v10.5
  Sets volume, balance, and mixing settings for a specified part of a tune.

`TuneSetPartTranspose` (page 1756) Deprecated in Mac OS X v10.5
  Modifies the pitch and volume of every note of a tune.

`TuneSetSofter` (page 1757) Deprecated in Mac OS X v10.5
  Adjusts the volume a tune is played at to the softer volume produced by QuickTime 2.1.

`TuneSetSoundLocalization` (page 1758) Deprecated in Mac OS X v10.5
  Passes sound localization data to a tune player.

`TuneSetTimeScale` (page 1758) Deprecated in Mac OS X v10.5
  Sets the time scale used by the specified tune player instance.

`TuneSetVolume` (page 1759) Deprecated in Mac OS X v10.5
  Sets the volume for an entire sequence.

`TuneStop` (page 1759) Deprecated in Mac OS X v10.5
Stops a currently playing sequence.

`TuneTask` (page 1760) Deprecated in Mac OS X v10.5
Lets a tune player to perform tasks it must perform at foreground task time.

`TuneUnroll` (page 1760) Deprecated in Mac OS X v10.5
Releases any note channel resources that may have been locked down by previous calls to TunePreroll for this tune player.

Supporting Functions

`DisposeMusicMIDISendUPP` (page 1676) Deprecated in Mac OS X v10.5
Disposes of a MusicMIDISendUPP pointer.

`DisposeMusicOfflineDataUPP` (page 1676) Deprecated in Mac OS X v10.5
Disposes of a MusicOfflineDataUPP pointer.

`DisposeTuneCallBackUPP` (page 1677) Deprecated in Mac OS X v10.5
Disposes of a TuneCallBackUPP pointer.

`DisposeTunePlayCallBackUPP` (page 1677) Deprecated in Mac OS X v10.5
Disposes of a TunePlayCallBackUPP pointer.

`MusicDerivedCloseResFile` (page 1678) Deprecated in Mac OS X v10.5
Closes a music movie resource file.

`MusicDerivedMIDISend` (page 1678) Deprecated in Mac OS X v10.5
Sends a MIDI packet to a music component.

`MusicDerivedOpenResFile` (page 1679) Deprecated in Mac OS X v10.5
Opens the music resource file for a music component.

`MusicDerivedSetPartInstrumentNumber` (page 1682) Deprecated in Mac OS X v10.5
Sets the instrument specified in the GCPart structure.

`MusicDerivedStorePartInstrument` (page 1683) Deprecated in Mac OS X v10.5
Undocumented

`MusicGenericGetKnobList` (page 1686) Deprecated in Mac OS X v10.5
Gets a list of the knobs of a given type for the generic music component.

`MusicGenericGetPart` (page 1687) Deprecated in Mac OS X v10.5
Gets a part used by the generic music component.

`MusicGenericSetResourceNumbers` (page 1687) Deprecated in Mac OS X v10.5
Undocumented

`MusicGetDrumNames` (page 1690) Deprecated in Mac OS X v10.5
Undocumented

`MusicGetInfoText` (page 1691) Deprecated in Mac OS X v10.5
Undocumented

`MusicGetInstrumentNames` (page 1693) Deprecated in Mac OS X v10.5
Undocumented

`NewMusicMIDISendUPP` (page 1741) Deprecated in Mac OS X v10.5
Allocates a Universal Procedure Pointer for the MusicMIDISendProc callback.

`NewMusicOfflineDataUPP` (page 1742) Deprecated in Mac OS X v10.5
Allocates a Universal Procedure Pointer for the MusicOfflineDataProc callback.

`NewTuneCallBackUPP` (page 1742) Deprecated in Mac OS X v10.5
    Allocates a Universal Procedure Pointer for the TuneCallBackProc callback.

`NewTunePlayCallBackUPP` (page 1743) Deprecated in Mac OS X v10.5
    Allocates a Universal Procedure Pointer for the TunePlayCallBackProc callback.

Functions

DisposeMusicMIDISendUPP

Disposes of a MusicMIDISendUPP pointer. (Deprecated in Mac OS X v10.5.)

```
void DisposeMusicMIDISendUPP (
   MusicMIDISendUPP userUPP
);
```

Parameters

userUPP

A `MusicMIDISendUPP` pointer. See `Universal Procedure Pointers`.

Return Value

You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes

Introduced in QuickTime 4.1.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`QuickTimeMusic.h`

DisposeMusicOfflineDataUPP

Disposes of a MusicOfflineDataUPP pointer. (Deprecated in Mac OS X v10.5.)

```
void DisposeMusicOfflineDataUPP (
   MusicOfflineDataUPP userUPP
);
```

Parameters

userUPP

A `MusicOfflineDataUPP` pointer. See `Universal Procedure Pointers`.

Return Value

You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes

Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In
`QuickTimeMusic.h`

DisposeTuneCallBackUPP

Disposes of a TuneCallBackUPP pointer. (Deprecated in Mac OS X v10.5.)

```
void DisposeTuneCallBackUPP (
   TuneCallBackUPP userUPP
);
```

Parameters

`userUPP`

A `TuneCallBackUPP` pointer. See `Universal Procedure Pointers`.

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In
`QuickTimeMusic.h`

DisposeTunePlayCallBackUPP

Disposes of a TunePlayCallBackUPP pointer. (Deprecated in Mac OS X v10.5.)

```
void DisposeTunePlayCallBackUPP (
   TunePlayCallBackUPP userUPP
);
```

Parameters

`userUPP`

A `TunePlayCallBackUPP` pointer. See `Universal Procedure Pointers`.

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In
`QuickTimeMusic.h`

MusicDerivedCloseResFile

Closes a music movie resource file. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicDerivedCloseResFile (
   MusicComponent mc,
   short resRefNum
);
```

Parameters

mc

A music component. Your software obtains this reference when calling `OpenComponent` or `OpenDefaultComponent`.

resRefNum

The resource file to be closed. Your application obtains this value from the `OpenMovieFile` (page 1416) function.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`QuickTimeMusic.h`

MusicDerivedMIDISend

Sends a MIDI packet to a music component. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicDerivedMIDISend (
   MusicComponent mc,
   MusicMIDIPacket *packet
);
```

Parameters

mc

A music component. Your software obtains this reference when calling `OpenComponent` or `OpenDefaultComponent` function.

packet

A pointer to the music MIDI packet to be sent.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`QuickTimeMusic.h`

MusicDerivedOpenResFile

Opens the music resource file for a music component. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicDerivedOpenResFile (
   MusicComponent mc
);
```

Parameters

mc

A music component. Your software obtains this reference when calling `OpenComponent` or `OpenDefaultComponent`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`QuickTimeMusic.h`

MusicDerivedSetInstrument

The complete instrument defined by the Part structure to the synthesizer. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicDerivedSetInstrument (
   MusicComponent mc,
   long partNumber,
   GCPart *p
);
```

Parameters

mc

The instance of the generic music component. Your software obtains this reference when calling `OpenComponent` or `OpenDefaultComponent`.

partNumber

The number of the part for this operation.

p

A pointer to the part for this operation.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In
`QuickTimeMusic.h`

MusicDerivedSetKnob

Called when any of the synthesizer's knobs are altered. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicDerivedSetKnob (
   MusicComponent mc,
   long knobType,
   long knobNumber,
   long knobValue,
   long partNumber,
   GCPart *p,
   GenericKnobDescription *gkd
);
```

Parameters

mc

The instance of the generic music component. Your software obtains this reference when calling `OpenComponent` or `OpenDefaultComponent`.

knobType

The type of knob that has been altered (see below). See these constants:

`kGenericMusicKnob`

`kGenericMusicInstrumentKnob`

`kGenericMusicDrumKnob`

knobNumber

The number of the knob that has been altered.

knobValue

The new value of the altered knob.

partNumber

The number of the part whose knob has been altered.

p

A pointer to the part whose knob has been altered.

gkd

A `GenericKnobDescription` structure for the knob.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function is called when any knob on the synthesizer is altered. It should look at the `GCPart` and the `GenericKnobDescription` structures and address the synthesizer hardware appropriately to set the new knob value. For a MIDI device, this means to construct a system-exclusive MIDI packet and send it to the MIDI routine received by the `MusicDerivedSetMIDI` (page 1681) call.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`QuickTimeMusic.h`

MusicDerivedSetMIDI

Sets the MIDI channel and other MIDI settings for MIDI output only. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicDerivedSetMIDI (
   MusicComponent mc,
   MusicMIDISendUPP midiProc,
   long refcon,
   long midiChannel
);
```

Parameters

mc
The instance of the generic music component. Your software obtains this reference when calling `OpenComponent` or `OpenDefaultComponent`.

midiProc
A pointer to the `MusicMIDISendProc` callback in your music component for performing MIDI output.

refcon
A reference constant sent to the callback specified by the `midiProc` parameter. Use this parameter to point to a data structure containing any information your callback needs.

midiChannel
The MIDI channel to use for the operation.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
A derived component for a MIDI synthesizer receives this call soon after it is opened. It should store the `midiProc`, `refCon`, and `midiChannel` parameters in its global variables. When the derived component needs to communicate with the synthesizer, it calls your `MusicMIDISendProc` function with this reference constant. The `midiChannel` variable specifies the "system channel" of the device.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`QuickTimeMusic.h`

MusicDerivedSetPart

Sets the polyphony for the part specified in the GCPart structure. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicDerivedSetPart (
   MusicComponent mc,
   long partNumber,
   GCPart *p
);
```

Parameters

mc
The instance of the generic music component. Your software obtains this reference when calling `OpenComponent` or `OpenDefaultComponent`.

partNumber
The number of the part for this operation.

p
A pointer to the part for this operation.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`QuickTimeMusic.h`

MusicDerivedSetPartInstrumentNumber

Sets the instrument specified in the GCPart structure. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicDerivedSetPartInstrumentNumber (
   MusicComponent mc,
   long partNumber,
   GCPart *p
);
```

Parameters

mc

A music component. Your software obtains this reference when calling `OpenComponent` or `OpenDefaultComponent`.

partNumber

The number of the part for this operation.

p

A pointer to the part for this operation.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`QuickTimeMusic.h`

MusicDerivedStorePartInstrument

Undocumented (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicDerivedStorePartInstrument (
   MusicComponent mc,
   long partNumber,
   GCPart *p,
   long instrumentNumber
);
```

Parameters

mc

An instance of the music component. Your software obtains this reference when calling `OpenComponent` or `OpenDefaultComponent`.

partNumber

The number of the part for this operation.

p

A pointer to the part for this operation.

instrumentNumber

Number of the instrument for this part. You can use `MusicFindTone` (page 1684) to get an instrument number.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In
`QuickTimeMusic.h`

MusicFindTone

Returns the number of the best-matching instrument provided by a specified music component. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicFindTone (
   MusicComponent mc,
   ToneDescription *td,
   long *libraryIndexOut,
   unsigned long *fit
);
```

Parameters

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (page 1722).

td

Pointer to a `ToneDescription` structure.

libraryIndexOut

On return, contains the number of the best-matching instrument. Only General MIDI numbers are guaranteed to be the same for later instantiations of the component.

fit

On return, a constant (see below) that indicates how well an instrument matches the tone description. See these constants:

`kInstrumentMatchSynthesizerType`

`kInstrumentMatchSynthesizerName`

`kInstrumentMatchName`

`kInstrumentMatchNumber`

`kInstrumentMatchGMNumber`

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
The music component searches for an instrument as follows:

If the `synthesizerType` field of the `td` parameter matches the type of the specified music component, it first tries to find an instrument that matches the value of the `instrumentNumber` field of the `td` parameter. If this value is in the range 129-16512, which specifies a GS instrument, and the GS instrument is not available, it tries to find the General MIDI instrument that corresponds to it, which has the number `((GSinstrumentnumber - 1) & 0x7F)` + 1. If the value is greater than 16512, which specifies a transient

ROM instrument or internal instrument index value, it tries to find an instrument that matches the `synthesizerName` field of the `td` parameter. If that fails, it tries to find an instrument that matches the value of the `gmNumber` field of the `td` parameter.

If the `synthesizerType` field of the `td` parameter does not match the type of the specified music component, it tries to find an instrument that matches the value of the `gmNumber` field of the `td` parameter.

If none of these rules apply, or the fields are blank (0 for the `type` or numeric fields, or zero-length for the strings), then the call returns instrument 1 and a fit parameter of zero.

The `synthesizerName` field may be ignored by the component; it is used by the note allocator when deciding which music device to use.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Related Sample Code
QTMusicToo

Declared In
`QuickTimeMusic.h`

MusicGenericConfigure

Informs the generic music component what services your music component requires and points to any resources that are necessary. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicGenericConfigure (
   MusicComponent mc,
   long mode,
   long flags,
   long baseResID
);
```

Parameters

`mc`
The instance of the generic music component. Your software obtains this reference when calling the Component Manager's `OpenComponent` or `OpenDefaultComponent` function.

`mode`
Must be 0.

flags
Flags (see below) that control the importation of MIDI files. See these constants:
`kGenericMusicDoMIDI`
`kGenericMusicBank0`
`kGenericMusicBank32`
`kGenericMusicErsatzMIDI`
`kGenericMusicCallKnobs`
`kGenericMusicCallParts`
`kGenericMusicCallInstrument`
`kGenericMusicCallNumber`
`kGenericMusicCallROMInstrument`

baseResID
The resource ID of the lowest-numbered resource used by your music component.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
The `baseResID` parameter is the lowest resource ID used by your component for the standard resources described above. Since the resource numbers are relative to this, you can include several music components in a single system extension.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`QuickTimeMusic.h`

MusicGenericGetKnobList

Gets a list of the knobs of a given type for the generic music component. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicGenericGetKnobList (
   MusicComponent mc,
   long knobType,
   GenericKnobDescriptionListHandle *gkdlH
);
```

Parameters

mc
The instance of the generic music component. Your software obtains this reference when calling `OpenComponent` or `OpenDefaultComponent`.

knobType
A constant (see below) that defines the type of knob. See these constants:
`kGenericMusicKnob`
`kGenericMusicInstrumentKnob`
`kGenericMusicDrumKnob`

gkdlH
: On return, a pointer to a handle to a `GenericKnobDescriptionList` structure.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In
`QuickTimeMusic.h`

MusicGenericGetPart

Gets a part used by the generic music component. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicGenericGetPart (
   MusicComponent mc,
   long partNumber,
   GCPart **part
);
```

Parameters

mc
: The instance of the generic music component. Your software obtains this reference when calling `OpenComponent` or `OpenDefaultComponent`.

partNumber
: The number of the part for this operation.

part
: A handle to a `GCPart` structure.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In
`QuickTimeMusic.h`

MusicGenericSetResourceNumbers

Undocumented (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicGenericSetResourceNumbers (
   MusicComponent mc,
   Handle resourceIDH
);
```

Parameters

mc

The instance of the generic music component. Your software obtains this reference when calling `OpenComponent` or `OpenDefaultComponent`.

resourceIDH

A handle to a resource ID.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`QuickTimeMusic.h`

MusicGetDescription

Returns a structure describing the synthesizer controlled by the music component device. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicGetDescription (
   MusicComponent mc,
   SynthesizerDescription *sd
);
```

Parameters

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (page 1722).

sd

Pointer to a `SynthesizerDescription` structure.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Related Sample Code

QTMusicToo

Declared In
`QuickTimeMusic.h`

MusicGetDeviceConnection

Determines how many hardware synthesizers are available to a music component and gets the IDs for those devices. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicGetDeviceConnection (
   MusicComponent mc,
   long index,
   long *id1,
   long *id2
);
```

Parameters

mc

Music component returned by `NAGetRegisteredMusicDevice` (page 1722).

index

Index of the device for which you want to find out the IDs. Set to 0 if you are calling to get the number of hardware devices.

id1

On return, a hardware synthesizer ID.

id2

On return, another hardware synthesizer ID.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
To get the number of hardware synthesizers available to the music component specified in the `mc` parameter and an index you can use to request ID numbers for a specific device, call this function with a value of 0 for the `index` parameter. You can then pass an index value in the `index` parameter, and the function returns hardware synthesizer IDs in the id1 and id2 parameters.

Special Considerations

This function is implemented only for hardware synthesizers, such as PCI card devices.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In
`QuickTimeMusic.h`

MusicGetDrumKnobDescription

Returns a description of a drum kit knob. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicGetDrumKnobDescription (
   MusicComponent mc,
   long knobIndex,
   KnobDescription *mkd
);
```

Parameters

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (page 1722).

knobIndex

A knob index or knob ID.

mkd

A pointer to a `KnobDescription` structure.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`QuickTimeMusic.h`

MusicGetDrumNames

Undocumented (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicGetDrumNames (
   MusicComponent mc,
   long modifiableInstruments,
   Handle *instrumentNumbers,
   Handle *instrumentNames
);
```

Parameters

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (page 1722).

modifiableInstruments

Undocumented

instrumentNumbers

Undocumented

instrumentNames

A pointer to a handle to the requested list of instrument name strings, formatted as a short integer followed by packed strings.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`QuickTimeMusic.h`

MusicGetInfoText

Undocumented (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicGetInfoText (
   MusicComponent mc,
   long selector,
   Handle *textH,
   Handle *styleH
);
```

Parameters

mc
: Music component instance identifier returned by `NAGetRegisteredMusicDevice` (page 1722).

selector
: *Undocumented*

textH
: *Undocumented*

styleH
: *Undocumented*

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`QuickTimeMusic.h`

MusicGetInstrumentAboutInfo

Obtains the information about an instrument that appears in its About box. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicGetInstrumentAboutInfo (
   MusicComponent mc,
   long part,
   InstrumentAboutInfo *iai
);
```

Parameters

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (page 1722).

part

Number of the part containing the instrument for which you want information.

iai

On return, a pointer to an `InstrumentAboutInfo` structure for the instrument currently on the specified synthesizer part.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`QuickTimeMusic.h`

MusicGetInstrumentInfo

Obtains a list of instruments supported by a synthesizer. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicGetInstrumentInfo (
   MusicComponent mc,
   long getInstrumentInfoFlags,
   InstrumentInfoListHandle *infoListH
);
```

Parameters

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (page 1722).

getInstrumentInfoFlags

Flags (see below) that specify limits to the list of instruments. See these constants:

`kGetInstrumentInfoNoBuiltIn`

`kGetInstrumentInfoMidiUserInst`

`kGetInstrumentInfoNoIText`

infoListH

On return, a pointer to a handle to an `InstrumentInfoList` structure that contains the list of instruments. This handle must be disposed of by the caller.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`QuickTimeMusic.h`

MusicGetInstrumentKnobDescription

Obtains the description of an instrument knob. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicGetInstrumentKnobDescription (
   MusicComponent mc,
   long knobIndex,
   KnobDescription *mkd
);
```

Parameters

`mc`
Music component instance identifier returned by `NAGetRegisteredMusicDevice` (page 1722).

`knobIndex`
A knob index or knob ID.

`mkd`
On return, a `KnobDescription` structure.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Related Sample Code
QTMusicToo

Declared In
`QuickTimeMusic.h`

MusicGetInstrumentNames

Undocumented (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicGetInstrumentNames (
   MusicComponent mc,
   long modifiableInstruments,
   Handle *instrumentNames,
   Handle *instrumentCategoryLasts,
   Handle *instrumentCategoryNames
);
```

Parameters

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (page 1722).

modifiableInstruments

Undocumented

instrumentNames

A pointer to a handle to the requested list of instrument name strings, formatted as a short integer followed by packed strings.

instrumentCategoryLasts

A pointer to a handle to a group of short integers, the first of which contains the number of integers to follow.

instrumentCategoryNames

A pointer to a handle to the requested list of instrument category name strings, formatted as a short integer followed by packed strings.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`QuickTimeMusic.h`

MusicGetKnob

Returns the value of the specified global synthesizer knob. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicGetKnob (
   MusicComponent mc,
   long knobID
);
```

Parameters

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (page 1722).

knobID

Knob index or ID.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion
A global knob controls an aspect of the entire synthesizer. It is not specific to a part within the synthesizer.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Related Sample Code
QTMusicToo

Declared In
`QuickTimeMusic.h`

MusicGetKnobDescription

Returns a pointer to an initialized knob description structure describing a global synthesizer knob. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicGetKnobDescription (
   MusicComponent mc,
   long knobIndex,
   KnobDescription *mkd
);
```

Parameters

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (page 1722).

knobIndex

Knob index or ID.

mkd

Pointer to a `KnobDescription` structure. The initialized structure provides default values associated with the particular knob.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
A global knob controls an aspect of the entire synthesizer; it is not limited to a part within the synthesizer. You can use the information returned by a call to this function to reset a knob to some known, usable value.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Related Sample Code
QTMusicToo

Declared In
`QuickTimeMusic.h`

MusicGetKnobSettingStrings

Returns a list of knob setting names known by the specified music component. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicGetKnobSettingStrings (
   MusicComponent mc,
   long knobIndex,
   long isGlobal,
   Handle *settingsNames,
   Handle *settingsCategoryLasts,
   Handle *settingsCategoryNames
);
```

Parameters

mc
Music component instance identifier returned by `NAGetRegisteredMusicDevice` (page 1722).

knobIndex
The knob index or knob ID.

isGlobal
If a knob index is used, indicates whether the specified knob is a global knob.

settingsNames
The requested list of knob setting strings formatted as a short followed by packed strings.

settingsCategoryLasts
A group of short integers, the first of which contains the number of shorts to follow.

settingsCategoryNames
Knob setting category names formatted as a short integer followed by a list of names.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
All handles must be disposed of by the caller.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`QuickTimeMusic.h`

MusicGetMasterTune

Returns the synthesizer's master tuning as a fixed-point value in semitones. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicGetMasterTune (
   MusicComponent mc
);
```

Parameters

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (page 1722).

Return Value

A fixed-point value representing the synthesizer's master tuning. The value is a fixed 16.16 number, allowing shifts by fractional values.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Related Sample Code

QTMusicToo

Declared In

`QuickTimeMusic.h`

MusicGetMIDIPorts

Returns the number of input and output ports a MIDI device has. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicGetMIDIPorts (
   MusicComponent mc,
   long *inputPortCount,
   long *outputPortCount
);
```

Parameters

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (page 1722).

inputPortCount

On return, the number of input MIDI ports available to the music component.

outputPortCount

On return, the number of output MIDI ports available to the music component.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Special Considerations

This call is implemented only for hardware synthesizers, such as PCI card devices.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In
`QuickTimeMusic.h`

MusicGetMIDIProc

Returns a pointer to the procedure a music component is using to process external MIDI notes. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicGetMIDIProc (
   MusicComponent mc,
   MusicMIDISendUPP *midiSendProc,
   long *refCon
);
```

Parameters

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (page 1722).

midiSendProc

Pointer to a MIDI serial port `MusicMIDISendProc` callback that processes external MIDI notes. This function was set by a previous call to `MusicSetMIDIProc` (page 1706). If no function has been set with `MusicSetMIDIProc`, this parameter returns 0.

refCon

A reference constant. The Movie Toolbox passes this reference constant to your `MusicMIDISendProc` each time it calls it. Use this parameter to point to a data structure containing any information your callback needs.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In
`QuickTimeMusic.h`

MusicGetPart

Returns the MIDI channel and maximum polyphony for a particular part. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicGetPart (
   MusicComponent mc,
   long part,
   long *midiChannel,
   long *polyphony
);
```

Parameters

mc
: Music component instance identifier returned by `NAGetRegisteredMusicDevice` (page 1722).

part
: The music component part requested.

midiChannel
: On return, a pointer to a MIDI channel. For non-MIDI devices, the MIDI channel pointed to by this parameter is 0.

polyphony
: On return, a pointer to the maximum number of voices or polyphony for the part..

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`QuickTimeMusic.h`

MusicGetPartAtomicInstrument

Returns the atomic instrument currently in a part. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicGetPartAtomicInstrument (
   MusicComponent mc,
   long part,
   AtomicInstrument *ai,
   long flags
);
```

Parameters

mc
: Music component instance identifier returned by `NAGetRegisteredMusicDevice` (page 1722).

part
: The part with the atomic instrument.

ai
: On return, an atomic instrument.

flags

A constant (see below) that specifies what pieces of information about an atomic instrument the caller is interested in. See these constants:

`kGetAtomicInstNoExpandedSamples`

`kGetAtomicInstNoOriginalSamples`

`kGetAtomicInstNoSamples`

`kGetAtomicInstNoKnobList`

`kGetAtomicInstNoInstrumentInfo`

`kGetAtomicInstOriginalKnobList`

`kGetAtomicInstAllKnobs`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`QuickTimeMusic.h`

MusicGetPartController

Returns the value of a specified controller on a specified part. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicGetPartController (
   MusicComponent mc,
   long part,
   MusicController controllerNumber
);
```

Parameters

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (page 1722).

part

Part whose controller value you want to get.

controllerNumber

On return, the controller number; see `Music Controllers`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In
`QuickTimeMusic.h`

MusicGetPartInstrumentNumber

Returns the instrument number currently assigned to a part. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicGetPartInstrumentNumber (
   MusicComponent mc,
   long part
);
```

Parameters

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (page 1722).

part

Part number containing the instrument.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In
`QuickTimeMusic.h`

MusicGetPartKnob

Retrieves the current value of a knob for a part. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicGetPartKnob (
   MusicComponent mc,
   long part,
   long knobID
);
```

Parameters

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (page 1722).

part

The part number.

knobID

The knob index or ID.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Related Sample Code
QTMusicToo

Declared In
`QuickTimeMusic.h`

MusicGetPartName

Returns the string name of a part. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicGetPartName (
   MusicComponent mc,
   long part,
   StringPtr name
);
```

Parameters

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (page 1722).

part

Part to get the name of.

name

On return, a string containing the part name.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
The name string is used by selection dialog boxes or configuration information.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Related Sample Code
QTMusicToo

Declared In
`QuickTimeMusic.h`

MusicPlayNote

Plays a note on a specified part at a specified pitch and velocity. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicPlayNote (
   MusicComponent mc,
   long part,
   long pitch,
   long velocity
);
```

Parameters

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (page 1722).

part

The part to play the note on.

pitch

The pitch at which to play the note. If the pitch is specified by a number from 0 to 127, it is a MIDI pitch, where 60 is middle C. If the pitch is a positive number above 65535, the value is a fixed-point pitch value. Thus, microtonal values may be specified.

velocity

How hard to strike the key. Values are 0-127 where 0 is silence. Velocity refers to how hard the key is struck (if performed on a keyboard-instrument); typically, this translates directly to volume, but on many synthesizers this also subtly alters the timbre of the tone.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The current note continues to play until a `MusicPlayNote` function with the same pitch and velocity of 0 turns the note off.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Related Sample Code

QTMusicToo

Declared In

`QuickTimeMusic.h`

MusicResetPart

Silences all sounds on a specified part and resets all controllers on that part to their default values. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicResetPart (
   MusicComponent mc,
   long part
);
```

Parameters

mc
Music component instance identifier returned by `NAGetRegisteredMusicDevice` (page 1722).

part
The number of the part.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
The default value to which controllers on the part are set is 0 for all controllers except volume. Volume is set to its maximum, 32767 (0x7FFF).

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`QuickTimeMusic.h`

MusicSendMIDI

Sends a MIDI packet to a specified port. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicSendMIDI (
   MusicComponent mc,
   long portIndex,
   MusicMIDIPacket *mp
);
```

Parameters

mc
Music component instance returned by `NAGetRegisteredMusicDevice` (page 1722).

portIndex
The index of the port to send the MIDI packet to. The index value is 1 through the port count returned by `MusicGetMIDIPorts` (page 1697).

mp
A pointer to the music MIDI packet to be sent. The function sends the MIDI music packet specified by this parameter to the specified port.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Special Considerations
This call is implemented only for hardware synthesizers, such as PCI card devices.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In
`QuickTimeMusic.h`

MusicSetKnob

Modifies the value of the specified global synthesizer knob. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicSetKnob (
   MusicComponent mc,
   long knobID,
   long knobValue
);
```

Parameters

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (page 1722).

knobID

Knob index or ID.

knobValue

Value for specified knob.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
A global knob controls an aspect of the entire synthesizer; it is not limited to a part within the synthesizer.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Related Sample Code
QTMusicToo

Declared In
`QuickTimeMusic.h`

MusicSetMasterTune

Alters a synthesizer's master tuning. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicSetMasterTune (
   MusicComponent mc,
   long masterTune
);
```

Parameters

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (page 1722).

masterTune

The amount by which to transpose the entire synthesizer in pitch. The value is a fixed 16.16 number, allowing shifts by fractional values.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Related Sample Code

QTMusicToo

Declared In

`QuickTimeMusic.h`

MusicSetMIDIProc

Informs the music component what procedure to call when it needs to send MIDI data. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicSetMIDIProc (
   MusicComponent mc,
   MusicMIDISendUPP midiSendProc,
   long refCon
);
```

Parameters

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (page 1722).

midiSendProc

A pointer to the `MusicMIDISendProc` callback to use when sending MIDI data.

refCon

A reference constant value. The Movie Toolbox passes this reference constant to your callback each time it calls it. Use this parameter to point to a data structure containing any information your callback needs.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This call is implemented only by music components for MIDI synthesizers.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`QuickTimeMusic.h`

MusicSetOfflineTimeTo

Advances the synthesizer clock when the synthesizer is not running in real time. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicSetOfflineTimeTo (
   MusicComponent mc,
   long newTimeStamp
);
```

Parameters

mc
: Music component instance identifier returned by `NAGetRegisteredMusicDevice` (page 1722).

newTimeStamp
: The number of samples to synthesize.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
The synthesizer may not be running in real time due to a call to `MusicStartOffline` (page 1713). Setting the time generates audio output from the synthesizer.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`QuickTimeMusic.h`

MusicSetPart

Sets the MIDI channel and maximum polyphony for a specified part. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicSetPart (
   MusicComponent mc,
   long part,
   long midiChannel,
   long polyphony
);
```

Parameters

mc
: Music component instance identifier returned by NAGetRegisteredMusicDevice (page 1722).

part
: Part whose MIDI channel and polyphony are to be set.

midiChannel
: The MIDI channel to set the part to. For non-MIDI devices, set this parameter to 0.

polyphony
: The maximum number of voices or polyphony for the part.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`QuickTimeMusic.h`

MusicSetPartAtomicInstrument

Initializes a part with an atomic instrument. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicSetPartAtomicInstrument (
   MusicComponent mc,
   long part,
   AtomicInstrumentPtr aiP,
   long flags
);
```

Parameters

mc
: Music component instance identifier returned by NAGetRegisteredMusicDevice (page 1722).

part
: The part to initialize with the atomic instrument to.

aiP
: The atomic instrument.

flags

Constants (see below) that specify details of initializing a part with an atomic instrument. See these constants:

`kGetAtomicInstNoExpandedSamples`

`kGetAtomicInstNoOriginalSamples`

`kGetAtomicInstNoSamples`

`kGetAtomicInstNoKnobList`

`kGetAtomicInstNoInstrumentInfo`

`kGetAtomicInstOriginalKnobList`

`kGetAtomicInstAllKnobs`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`QuickTimeMusic.h`

MusicSetPartController

Initializes the value of a specified controller on a specified part. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicSetPartController (
   MusicComponent mc,
   long part,
   MusicController controllerNumber,
   long controllerValue
);
```

Parameters

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (page 1722).

part

Part whose controller value you want to set.

controllerNumber

Controller number; see `Music Controllers`.

controllerValue

Value for controller.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`QuickTimeMusic.h`

MusicSetPartInstrumentNumber

Superseded by MusicSetPartInstrumentNumberInterruptSafe. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicSetPartInstrumentNumber (
   MusicComponent mc,
   long part,
   long instrumentNumber
);
```

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`QuickTimeMusic.h`

MusicSetPartInstrumentNumberInterruptSafe

Initializes a part with a particular instrument. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicSetPartInstrumentNumberInterruptSafe (
   MusicComponent mc,
   long part,
   long instrumentNumber
);
```

Parameters

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (page 1722).

part

Part to be initialized.

instrumentNumber

Number of instrument to initialize part with. You can use `MusicFindTone` (page 1684) to get an instrument number.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Special Considerations

You can call this function at interrupt time.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`QuickTimeMusic.h`

MusicSetPartKnob

Sets a knob for a specified part. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicSetPartKnob (
   MusicComponent mc,
   long part,
   long knobID,
   long knobValue
);
```

Parameters

mc
Music component instance identifier returned by `NAGetRegisteredMusicDevice` (page 1722).

part
The part number.

knobID
The index or ID of the knob to be set.

knobValue
The value to set the knob to.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Related Sample Code
QTMusicToo

Declared In
`QuickTimeMusic.h`

MusicSetPartName

Changes the name of an instrument in a specified part. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicSetPartName (
   MusicComponent mc,
   long part,
   StringPtr name
);
```

Parameters

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (page 1722).

part

Part to apply name to.

name

A pointer to the name to apply to part.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

You might want to change the name of a modified instrument before saving it. The instrument name string is used by selection dialog and configuration information boxes.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Related Sample Code

QTMusicToo

Declared In

`QuickTimeMusic.h`

MusicSetPartSoundLocalization

Passes sound localization data to a specified synthesizer part. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicSetPartSoundLocalization (
   MusicComponent mc,
   long part,
   Handle data
);
```

Parameters

mc

Music component instance identifier.

part

The part to pass the data to.

data

The sound localization data.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Use the functions described in this section to get and modify the master tuning of the synthesizer, to play off line, and to allow the music component to perform tasks it must perform at foreground task time.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`QuickTimeMusic.h`

MusicStartOffline

Informs the QuickTime music synthesizer that the music will not be played through the speakers. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicStartOffline (
   MusicComponent mc,
   unsigned long *numChannels,
   UnsignedFixed *sampleRate,
   unsigned short *sampleSize,
   MusicOfflineDataUPP dataProc,
   long dataProcRefCon
);
```

Parameters

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (page 1722).

numChannels

Number of channels in the music sample; 1 indicates monaural, 2 indicates stereo.

sampleRate

The number of samples per second.

sampleSize

The size of the music sample: 8-bit or 16-bit.

dataProc

A pointer to a `MusicOfflineDataProc` callback to handle the audio data.

dataProcRefCon

A reference constant to pass to the `MusicOfflineDataProc` callback. Use this parameter to point to a data structure containing any information your callback needs.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Audio data will be sent to a function that will create a sound file to be played back later. You pass this function the requested values for the `numChannels`, `sampleRate`, and `sampleSize` parameters. When the function returns, those parameters contain the actual values used.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`QuickTimeMusic.h`

MusicStorePartInstrument

Puts whatever instrument is on the specified part into the synthesizer's instrument store. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicStorePartInstrument (
   MusicComponent mc,
   long part,
   long instrumentNumber
);
```

Parameters

mc
Music component instance identifier returned by `NAGetRegisteredMusicDevice` (page 1722).

part
Part containing the instrument to be stored.

instrumentNumber
Instrument number at which to store the part. The value must be between 1 and the synthesizer's modifiable instrument count, as defined by the `modifiableInstrumentCount` field of the synthesizer's `SynthesizerDescription` structure.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function lets you store modified instruments.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`QuickTimeMusic.h`

MusicTask

Allows a music component to perform tasks it must perform at foreground task time. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicTask (
   MusicComponent mc
);
```

Parameters

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (page 1722).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function must be called periodically. In the case of the QuickTime music synthesizer, instruments cannot be loaded from disk at interrupt time, so if the `NASetInstrumentNumberInterruptSafe` (page 1735) function is called, the instrument is loaded during the next `MusicTask` call.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`QuickTimeMusic.h`

MusicUseDeviceConnection

Tells a music component which hardware synthesizer to talk to. (Deprecated in Mac OS X v10.5.)

```
ComponentResult MusicUseDeviceConnection (
   MusicComponent mc,
   long id1,
   long id2
);
```

Parameters

mc

Music component instance identifier returned by `NAGetRegisteredMusicDevice` (page 1722).

id1

The ID of the device returned in the id1 parameter of `MusicGetDeviceConnection` (page 1689).

id2

The ID of the device returned in the id2 parameter of `MusicGetDeviceConnection` (page 1689).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Special Considerations

This call is implemented only for hardware synthesizers, such as PCI card devices.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In
`QuickTimeMusic.h`

NACopyrightDialog

Displays a copyright dialog box with information specific to a music device. (Deprecated in Mac OS X v10.5.)

```
ComponentResult NACopyrightDialog (
   NoteAllocator na,
   PicHandle p,
   StringPtr author,
   StringPtr copyright,
   StringPtr other,
   StringPtr title,
   ModalFilterUPP filterProc,
   long refCon
);
```

Parameters

na
: You obtain the note allocator identifier by calling `OpenComponent`.

p
: A handle to a `Picture` structure containing the image resource for the dialog box.

author
: A pointer to a string containing author information.

copyright
: A pointer to a string containing copyright information.

other
: A pointer to a string containing any additional information.

title
: A pointer to a string containing title information.

filterProc
: Pointer to a `ModalFilterProc` callback.

refCon
: A reference constant value. The Movie Toolbox passes this reference constant to your `ModalFilterProc` each time it calls it. Use this parameter to point to a data structure containing any information your callback needs.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In
`QuickTimeMusic.h`

NADisposeNoteChannel

Deletes a specified note channel. (Deprecated in Mac OS X v10.5.)

```
ComponentResult NADisposeNoteChannel (
   NoteAllocator na,
   NoteChannel noteChannel
);
```

Parameters

na
: You obtain the note allocator identifier by calling `OpenComponent`.

noteChannel
: Note channel to be disposed. You obtain the note channel identifier from `NANewNoteChannel` (page 1724) or `NANewNoteChannelFromAtomicInstrument` (page 1724).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Related Sample Code
qtmusic
qtmusic.win
QTMusicToo

Declared In
`QuickTimeMusic.h`

NAFindNoteChannelTone

Locates the instrument that best fits a requested tone description for a specific channel. (Deprecated in Mac OS X v10.5.)

```
ComponentResult NAFindNoteChannelTone (
   NoteAllocator na,
   NoteChannel noteChannel,
   ToneDescription *td,
   long *instrumentNumber
);
```

Parameters

na
: You obtain the note allocator identifier by calling `OpenComponent`.

noteChannel
: The note channel for which you want an instrument. You obtain the note channel identifier from `NANewNoteChannel` (page 1724) or `NANewNoteChannelFromAtomicInstrument` (page 1724).

td
: A `ToneDescription` structure that describes the instrument fit.

instrumentNumber
: On return, the number of the instrument that best fits the tone description.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`QuickTimeMusic.h`

NAGetController

Retrieves the controller settings for a note channel. (Deprecated in Mac OS X v10.5.)

```
ComponentResult NAGetController (
   NoteAllocator na,
   NoteChannel noteChannel,
   long controllerNumber,
   long *controllerValue
);
```

Parameters

na
: You obtain the note allocator identifier by calling `OpenComponent`.

noteChannel
: Note channel for which to get controller settings. You obtain the note channel identifier from `NANewNoteChannel` (page 1724) or `NANewNoteChannelFromAtomicInstrument` (page 1724).

controllerNumber
: The controller for which to get settings; see `Music Controllers`.

controllerValue
: On return, the value for the controller setting, typically 0 (0x00.00) to 32767 (0x7F.FF).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In
`QuickTimeMusic.h`

NAGetIndNoteChannel

Returns the number of note channels handled by the specified note allocator instance. (Deprecated in Mac OS X v10.5.)

```
ComponentResult NAGetIndNoteChannel (
   NoteAllocator na,
   long index,
   NoteChannel *nc,
   long *seed
);
```

Parameters

na
: You obtain the note allocator identifier from the Component Manager's `OpenComponent` function.

index
: The index of the note channel. If 0, the result is still the number of note channels, but the `nc` parameter is not filled out.

nc
: The note channel requested.

seed
: A number that changes on successive calls if anything significant changes about a note channel; for example, if the note channel has been reallocated or released.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function can also return a requested note channel. To get a count of the note channels, pass 0 in the `index` parameter. To get a specific note channel, pass the index value returned by a previous call to `NAGetIndNoteChannel`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In
`QuickTimeMusic.h`

NAGetKnob

Obtains the value of a knob for a given note channel. (Deprecated in Mac OS X v10.5.)

```
ComponentResult NAGetKnob (
   NoteAllocator na,
   NoteChannel noteChannel,
   long knobNumber,
   long *knobValue
);
```

Parameters

na

You obtain the note allocator identifier by calling `OpenComponent`.

noteChannel

The note channel whose knob value you want to get. You obtain the note channel identifier from `NANewNoteChannel` (page 1724) or `NANewNoteChannelFromAtomicInstrument` (page 1724).

knobNumber

The index or ID of the knob whose value you want to get.

knobValue

On return, a pointer to the value of the knob.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`QuickTimeMusic.h`

NAGetMIDIPorts

The MIDI input and output ports available to a note allocator. (Deprecated in Mac OS X v10.5.)

```
ComponentResult NAGetMIDIPorts (
   NoteAllocator na,
   QTMIDIPortListHandle *inputPorts,
   QTMIDIPortListHandle *outputPorts
);
```

Parameters

na

You obtain the note allocator identifier by calling `OpenComponent`.

inputPorts

On return, a handle giving the number of input ports (the first two bytes) followed by a list of `QTMIDIPort` structures.

outputPorts

On return, a handle giving the number of output ports (the first two bytes) followed by a list of `QTMIDIPort` structures.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This routine calls the QuickTime MIDI components to query them.

Special Considerations

`NAGetMIDIPorts` is the correct call for applications to make. They should not call `QTMIDIGetMIDIPorts`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In
`QuickTimeMusic.h`

NAGetNoteChannelInfo

Returns the index of the music component for the allocated channel and its part number on that music component. (Deprecated in Mac OS X v10.5.)

```
ComponentResult NAGetNoteChannelInfo (
   NoteAllocator na,
   NoteChannel noteChannel,
   long *index,
   long *part
);
```

Parameters

na

You obtain the note allocator identifier by calling `OpenComponent`.

noteChannel

Note channel to get information about. You obtain the note channel identifier from `NANewNoteChannel` (page 1724) or `NANewNoteChannelFromAtomicInstrument` (page 1724).

index

Music component index.

part

Music component part pointer.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
The `NAGetNoteChannelInfo` function allows direct access to the music component allocated to the note channel by the note allocator. The index returned becomes invalid if music components are subsequently registered or unregistered.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Related Sample Code
QTMusicToo

Declared In
`QuickTimeMusic.h`

NAGetNoteRequest

Retrieves the NoteRequest structure that was passed to a note channel. (Deprecated in Mac OS X v10.5.)

```
ComponentResult NAGetNoteRequest (
   NoteAllocator na,
   NoteChannel noteChannel,
   NoteRequest *nrOut
);
```

Parameters

na
: You obtain the note allocator identifier from the Component Manager's `OpenComponent` function.

noteChannel
: The note channel whose note request you want to get. You obtain the note channel identifier from `NANewNoteChannel` (page 1724) or `NANewNoteChannelFromAtomicInstrument` (page 1724).

nrOut
: On return, the `NoteRequest` structure that was used when the specified note channel was allocated.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`QuickTimeMusic.h`

NAGetRegisteredMusicDevice

Returns details about music components registered to the specified note allocator instance. (Deprecated in Mac OS X v10.5.)

```
ComponentResult NAGetRegisteredMusicDevice (
   NoteAllocator na,
   long index,
   OSType *synthType,
   Str31 name,
   SynthesizerConnections *connections,
   MusicComponent *mc
);
```

Parameters

na

You obtain the note allocator identifier from the Component Manager's `OpenComponent` function.

index

The index of the music component to get information about. To get a count of the registered music components, pass 0 in the `index` parameter. The return value is the count of components. To get information about one of the music components registered with the note allocator, pass the music component index in the `index` parameter. The index value can be 1 through the number of registered components returned by a previous call to `NAGetRegisteredMusicDevice`.

synthType

Synthesizer type.

name

Synthesizer name as a text string.

connections

A synthesizer connections for MIDI devices structure.

mc

Music component instance identifier.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
If you request information about a specific registered music component, this function returns the type of synthesizer the component supports in the `synthType` parameter, the name of the synthesizer in the name parameter, and the music component identifier in the `mc` parameter. For MIDI devices, it returns a pointer to a MIDI devices structure with information about the synthesizer connections.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Related Sample Code
QTMusicToo

Declared In
`QuickTimeMusic.h`

NANewNoteChannel

Requests a new note channel with the qualities described in a NoteRequest structure. (Deprecated in Mac OS X v10.5.)

```
ComponentResult NANewNoteChannel (
   NoteAllocator na,
   NoteRequest *noteRequest,
   NoteChannel *outChannel
);
```

Parameters

na

You obtain the note allocator identifier from the Component Manager's `OpenComponent` function.

noteRequest

A pointer to a `NoteRequest` structure.

outChannel

On return, a pointer to an identifier for a new note channel or `NIL` if the function fails to create a note channel.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function searches all available music components for the instrument that best matches the specifications in the `ToneDescription` structure that is contained within the `noteRequest` parameter. If an error occurs, the new note channel is initialized to `NIL`. The caller can request an instrument that is not currently allocated to a part. In that case, this function may return a value in `outChannel`, even though the request cannot initially be satisfied. The note channel may become valid at a later time, as other note channels are released or other music components are registered.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Related Sample Code

qtmusic

qtmusic.win

QTMusicToo

Declared In

`QuickTimeMusic.h`

NANewNoteChannelFromAtomicInstrument

Requests a new note channel for an atomic instrument. (Deprecated in Mac OS X v10.5.)

```
ComponentResult NANewNoteChannelFromAtomicInstrument (
   NoteAllocator na,
   AtomicInstrumentPtr instrument,
   long flags,
   NoteChannel *outChannel
);
```

Parameters

na

You obtain the note allocator identifier from the Component Manager's `OpenComponent` function.

instrument

A pointer to the atomic instrument. This may be a dereferenced locked QT atom container.

flags

Flags (see below) that specify details of initializing a part with an atomic instrument. See these constants:

`kSetAtomicInstKeepOriginalInstrument`

`kSetAtomicInstShareAcrossParts`

`kSetAtomicInstCallerTosses`

`kSetAtomicInstDontPreprocess`

outChannel

On return, a pointer to an identifier for a new note channel or `NIL` if the function fails to create a note channel.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function takes a note allocator identifier in the `na` parameter and a pointer to the atomic instrument you are requesting a new channel for in the `instrument` parameter. Among other things, you can specify how to handle the expanded sample with the `flags` parameter. The function returns the note channel allocated for the instrument in the `outChannel` parameter, or `NIL` if an error occurs.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Related Sample Code

qtmusic

qtmusic.win

Declared In

`QuickTimeMusic.h`

NAPickArrangement

Displays a dialog box to allow instrument selection. (Deprecated in Mac OS X v10.5.)

```
ComponentResult NAPickArrangement (
   NoteAllocator na,
   ModalFilterUPP filterProc,
   StringPtr prompt,
   long zero1,
   long zero2,
   Track t,
   StringPtr songName
);
```

Parameters

na
You obtain the note allocator identifier by calling `OpenComponent`.

filterProc
A Universal Procedure Pointer to a `ModalFilterProc` callback.

prompt
A pointer to a dialog box prompt string.

zero1
Must be 0.

zero2
Must be 0.

t
The arrangement movie track number.

songName
A pointer to the name of a song to display in the dialog box.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`QuickTimeMusic.h`

NAPickEditInstrument

Presents a user interface for changing the instrument in a live note channel or modifying an atomic instrument. (Deprecated in Mac OS X v10.5.)

```
ComponentResult NAPickEditInstrument (
   NoteAllocator na,
   ModalFilterUPP filterProc,
   StringPtr prompt,
   long refCon,
   NoteChannel nc,
   AtomicInstrument ai,
   long flags
);
```

Parameters

na

You obtain the note allocator identifier by calling `OpenComponent`.

filterProc

Pointer to a `ModalFilterProc` callback.

prompt

Dialog box prompt "New Instrument".

refCon

A reference constant value. The Movie Toolbox passes this reference constant to your `ModalFilterProc` callback each time it calls it. Use this parameter to point to a data structure containing any information your callback needs.

nc

The live note channel that appears in the dialog box. If you specify a note channel, set the `ai` parameter to 0. You obtain the note channel identifier from `NANewNoteChannel` (page 1724) or `NANewNoteChannelFromAtomicInstrument` (page 1724).

ai

The atomic instrument that appears in the dialog box. If you specify an atomic instrument, set the `nc` parameter to 0.

flags

Flags (see below) that limit the instruments presented. If the `kPickDontMix` flag is set, the dialog box does not display a mix of synthesizer part types. For example, if the current instrument is a drum, only available drums appear in the dialog box. The `kPickSameSynth` flag allows selections only within the current synthesizer. The `kPickUserInsts` flag allows user modifiable instruments to appear. If the `kPickEditAllowPick` flag is not set, no dialog box appears. See these constants:

`kPickDontMix`

`kPickSameSynth`

`kPickUserInsts`

`kPickEditAllowPick`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`QuickTimeMusic.h`

NAPickInstrument

Presents a user interface for picking an instrument. (Deprecated in Mac OS X v10.5.)

```
ComponentResult NAPickInstrument (
   NoteAllocator na,
   ModalFilterUPP filterProc,
   StringPtr prompt,
   ToneDescription *sd,
   unsigned long flags,
   long refCon,
   long reserved1,
   long reserved2
);
```

Parameters

na

You obtain the note allocator identifier by calling `OpenComponent`.

filterProc

Pointer to a `ModalFilterProc` callback.

prompt

A pointer to the dialog box prompt "New Instrument".

sd

On entry, the tone description of the instrument that appears in the picker dialog box. On return, a tone description of the instrument the user selected.

flags

Flags (see below) that determine whether to display the picker dialog box and what instruments appear for selection. If the `kPickDontMix` flag is set, the dialog box does not display a mix of synthesizer part types. For example, if the current instrument is a drum, only available drums appear in the dialog box. The `kPickSameSynth` flag allows selections only within the current synthesizer. The `kPickUserInsts` flag allows user modifiable instruments to appear. The `kPickEditAllowPick` flag is used only with `NAPickEditInstrument` (page 1726). See these constants:

`kPickDontMix`

`kPickSameSynth`

`kPickUserInsts`

refCon

A reference constant value. The Movie Toolbox passes this reference constant to your `ModalFilterProc` callback each time it calls it. Use this parameter to point to a data structure containing any information your callback needs.

reserved1

Must contain 0.

reserved2

Must contain 0.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Related Sample Code
qtmusic
qtmusic.win
QTMusicToo

Declared In
`QuickTimeMusic.h`

NAPlayNote

Plays a note with a specified pitch and velocity on the specified note channel. (Deprecated in Mac OS X v10.5.)

```
ComponentResult NAPlayNote (
   NoteAllocator na,
   NoteChannel noteChannel,
   long pitch,
   long velocity
);
```

Parameters

na
: You obtain the note allocator identifier from `OpenComponent`.

noteChannel
: The note channel to play the note. You obtain the note channel identifier from the `NANewNoteChannel` (page 1724) or the `NANewNoteChannelFromAtomicInstrument` (page 1724) function.

pitch
: The pitch at which to play the note. You can specify values as integer pitch values (0-127 where 60 is middle C) or fractional pitch values (256 (0x1.00) through 32767 (0x7F.FF)). If the pitch is a number from 0 to 127, then it is the MIDI pitch, where 60 is middle C. If the pitch is a positive number above 65535, then the value is a fixed-point pitch value. Thus, microtonal values can be specified. Negative values are not defined and should not be used.

velocity
: The velocity with which the key is struck. Typically, this translates directly to volume, but on many synthesizers this also subtly alters the timbre of the tone. A value of 0 is silence; a value of 127 is maximum force.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Related Sample Code
qtmusic
qtmusic.win

QTMusicToo

Declared In
`QuickTimeMusic.h`

NAPrerollNoteChannel

Attempts to reallocate the note channel if it was invalid previously. (Deprecated in Mac OS X v10.5.)

```
ComponentResult NAPrerollNoteChannel (
   NoteAllocator na,
   NoteChannel noteChannel
);
```

Parameters

`na`
You obtain the note allocator identifier by calling `OpenComponent`.

`noteChannel`
Note channel to be re-allocated. You obtain the note channel identifier from `NANewNoteChannel` (page 1724) or `NANewNoteChannelFromAtomicInstrument` (page 1724).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
The `NAPrerollNoteChannel` function attempts to reallocate the note channel, if it was invalid previously. It could have been invalid if there were no available voices on any registered music components when the note channel was created.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Related Sample Code
QTMusicToo

Declared In
`QuickTimeMusic.h`

NARegisterMusicDevice

Registers a music component with the note allocator. (Deprecated in Mac OS X v10.5.)

```
ComponentResult NARegisterMusicDevice (
   NoteAllocator na,
   OSType synthType,
   Str31 name,
   SynthesizerConnections *connections
);
```

Parameters

na

You obtain the note allocator identifier from `OpenComponent`.

synthType

Subtype of the music component.

name

The synthesizer name. This parameter provides a means of distinguishing multiple instances of the same type of device and is a string that can be displayed to the user. If no value is passed in the name parameter, the name defaults to the name of the music component type. The name appears in the instrument picker dialog box.

connections

A `SynthesizerConnections` structure that describes the hardware connections to a MIDI device.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`QuickTimeMusic.h`

NAResetNoteChannel

Turns off all currently active notes on the note channel and resets all controllers to their default values. (Deprecated in Mac OS X v10.5.)

```
ComponentResult NAResetNoteChannel (
   NoteAllocator na,
   NoteChannel noteChannel
);
```

Parameters

na

You obtain the note allocator identifier by calling `OpenComponent`.

noteChannel

The note channel to reset. You obtain the note channel identifier from `NANewNoteChannel` (page 1724) or `NANewNoteChannelFromAtomicInstrument` (page 1724).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function resets the specified note channel by turning "off" any note currently playing. All controllers are reset to their default state. The effects of the `NAResetNoteChannel` call are propagated down to the allocated part within the appropriate music component.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Related Sample Code
QTMusicToo

Declared In
`QuickTimeMusic.h`

NASaveMusicConfiguration

Saves the current list of registered devices to a file. (Deprecated in Mac OS X v10.5.)

```
ComponentResult NASaveMusicConfiguration (
   NoteAllocator na
);
```

Parameters

na

You obtain the note allocator identifier by calling `OpenComponent`.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
The `NASaveMusicConfiguration` function saves the current list of registered devices to a file. This file is read whenever a note allocator connection is opened, restoring the previously configured list of devices. The list is saved in the QuickTime Preferences file.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`QuickTimeMusic.h`

NASendMIDI

Sends a MIDI music packet to a synthesizer that contains a specific note channel. (Deprecated in Mac OS X v10.5.)

```
ComponentResult NASendMIDI (
   NoteAllocator na,
   NoteChannel noteChannel,
   MusicMIDIPacket *mp
);
```

Parameters

na

You obtain the note allocator identifier from the Component Manager's `OpenComponent` function.

noteChannel

The function sends the packet to the synthesizer that contains this note channel. You obtain the note channel identifier from the `NANewNoteChannel` (page 1724) or the `NANewNoteChannelFromAtomicInstrument` (page 1724) function.

mp

The music packet to be sent.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function sends the MIDI music packet pointed to by the `mp` parameter to the synthesizer that contains the note channel identified by the `noteChannel` parameter. The `na` parameter specifies the note allocator instance to use.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`QuickTimeMusic.h`

NASetAtomicInstrument

Initializes a synthesizer part with an atomic instrument. (Deprecated in Mac OS X v10.5.)

```
ComponentResult NASetAtomicInstrument (
   NoteAllocator na,
   NoteChannel noteChannel,
   AtomicInstrumentPtr instrument,
   long flags
);
```

Parameters

na

You obtain the note allocator identifier by calling `OpenComponent`.

noteChannel

The note channel to apply the atomic instrument to. You obtain the note channel identifier from `NANewNoteChannel` (page 1724) or `NANewNoteChannelFromAtomicInstrument` (page 1724).

instrument

A pointer to the atomic instrument. This can be a locked, dereferenced atomic instrument.

flags

Flags (see below) that detail how to initialize the part. See these constants:

`kSetAtomicInstKeepOriginalInstrument`

`kSetAtomicInstShareAcrossParts`

`kSetAtomicInstCallerTosses`

`kSetAtomicInstDontPreprocess`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`QuickTimeMusic.h`

NASetController

Changes the controller setting on a note channel to a specified value. (Deprecated in Mac OS X v10.5.)

```
ComponentResult NASetController (
   NoteAllocator na,
   NoteChannel noteChannel,
   long controllerNumber,
   long controllerValue
);
```

Parameters

na

You obtain the note allocator identifier by calling `OpenComponent`.

noteChannel

Note channel on which to change controller. You obtain the note channel identifier from `NANewNoteChannel` (page 1724) or `NANewNoteChannelFromAtomicInstrument` (page 1724).

controllerNumber

The controller to set; see `Music Controllers`.

controllerValue

Value for controller setting; typically 0 (0x00.00) to 32767 (0x7F.FF).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Related Sample Code

qtmusic

qtmusic.win

QTMusicToo

Declared In

`QuickTimeMusic.h`

NASetInstrumentNumber

Initializes initializes a synthesizer part with the specified instrument. (Deprecated in Mac OS X v10.5.)

```
ComponentResult NASetInstrumentNumber (
   NoteAllocator na,
   NoteChannel noteChannel,
   long instrumentNumber
);
```

Parameters

na
: You obtain the note allocator identifier by calling `OpenComponent`.

noteChannel
: Note channel to initialize with the instrument. You obtain the note channel identifier from `NANewNoteChannel` (page 1724) or `NANewNoteChannelFromAtomicInstrument` (page 1724).

instrumentNumber
: Number of the instrument to initialize the part with. This number is unique to each synthesizer. General MIDI synthesizers all share the range 1-128 and 16365 to `kLastDrumKit`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`QuickTimeMusic.h`

NASetInstrumentNumberInterruptSafe

Initializes a synthesizer part with the specified instrument during interrupt time. (Deprecated in Mac OS X v10.5.)

```
ComponentResult NASetInstrumentNumberInterruptSafe (
   NoteAllocator na,
   NoteChannel noteChannel,
   long instrumentNumber
);
```

Parameters

na

You obtain the note allocator identifier by calling `OpenComponent`.

noteChannel

Note channel to initialize with the instrument. You obtain the note channel identifier from `NANewNoteChannel` (page 1724) or `NANewNoteChannelFromAtomicInstrument` (page 1724).

instrumentNumber

Number of the instrument to initialize the part with.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Special Considerations

If the instrument is not already loaded when you call `NASetInstrumentNumberInterruptSafe`, you have to wait for the next call to `NATask` (page 1740) for the instrument to become available.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`QuickTimeMusic.h`

NASetKnob

Sets a note channel knob to a particular value. (Deprecated in Mac OS X v10.5.)

```
ComponentResult NASetKnob (
   NoteAllocator na,
   NoteChannel noteChannel,
   long knobNumber,
   long knobValue
);
```

Parameters

na

You obtain the note allocator identifier by calling `OpenComponent`.

noteChannel

Note channel on which to set the knob value. You obtain the note channel identifier from `NANewNoteChannel` (page 1724) or `NANewNoteChannelFromAtomicInstrument` (page 1724).

knobNumber

Index or ID of the knob to be set.

knobValue
Value to set knob to.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Related Sample Code
QTMusicToo

Declared In
`QuickTimeMusic.h`

NASetNoteChannelBalance

Modifies the pan controller setting for a note channel. (Deprecated in Mac OS X v10.5.)

```
ComponentResult NASetNoteChannelBalance (
   NoteAllocator na,
   NoteChannel noteChannel,
   long balance
);
```

Parameters

na
You obtain the note allocator identifier by calling `OpenComponent`.

noteChannel
The note channel to be balanced. You obtain the note channel identifier from `NANewNoteChannel` (page 1724) or `NANewNoteChannelFromAtomicInstrument` (page 1724).

balance
Specifies how to modify the pan controller setting. Valid values are from -128 to 128 for left to right balance.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`QuickTimeMusic.h`

NASetNoteChannelSoundLocalization

Passes sound localization data to a note channel. (Deprecated in Mac OS X v10.5.)

```
ComponentResult NASetNoteChannelSoundLocalization (
   NoteAllocator na,
   NoteChannel noteChannel,
   Handle data
);
```

Parameters

na

You obtain the note allocator identifier by calling `OpenComponent`.

noteChannel

The note channel to pass the data to. You obtain the note channel identifier from `NANewNoteChannel` (page 1724) or `NANewNoteChannelFromAtomicInstrument` (page 1724).

data

Sound localization data.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`QuickTimeMusic.h`

NASetNoteChannelVolume

Sets the volume on the specified note channel. (Deprecated in Mac OS X v10.5.)

```
ComponentResult NASetNoteChannelVolume (
   NoteAllocator na,
   NoteChannel noteChannel,
   Fixed volume
);
```

Parameters

na

You obtain the note allocator identifier from the Component Manager's `OpenComponent` function.

noteChannel

The note channel to reset. You obtain the note channel identifier from the `NANewNoteChannel` (page 1724) or the `NANewNoteChannelFromAtomicInstrument` (page 1724) function.

volume

A fixed 16.16 number. `NASetNoteChannelVolume` sets the volume for the note channel, which is different from a `kControllerVolume` setting. Both volume settings allow fractional values of 0.0 to 1.0. Each value modifies the other. For example, a `kControllerVolume` value of 0.5 and a `NASetNoteChannelVolume` value of 0.5 result in a 0.25 volume level.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Related Sample Code
qtmusic
qtmusic.win

Declared In
`QuickTimeMusic.h`

NAStuffToneDescription

Initializes a tone description structure with the details of a General MIDI note channel. (Deprecated in Mac OS X v10.5.)

```
ComponentResult NAStuffToneDescription (
   NoteAllocator na,
   long gmNumber,
   ToneDescription *td
);
```

Parameters

na
: You obtain the note allocator identifier from the Component Manager's `OpenComponent` function.

gmNumber
: A General MIDI instrument number.

td
: On return, an initialized tone description. The instrument name field will be filled in with the string name for the instrument.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Related Sample Code
qtmusic
qtmusic.win
QTMusicToo

Declared In

`QuickTimeMusic.h`

NATask

Called periodically to allow the note allocator to perform tasks in foreground task time. (Deprecated in Mac OS X v10.5.)

```
ComponentResult NATask (
   NoteAllocator na
);
```

Parameters

na

You obtain the note allocator identifier from the Component Manager's `OpenComponent` function.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The `NATask` function calls each registered music component's `MusicTask` (page 1714) function.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`QuickTimeMusic.h`

NAUnregisterMusicDevice

Removes a previously registered music component from the note allocator. (Deprecated in Mac OS X v10.5.)

```
ComponentResult NAUnregisterMusicDevice (
   NoteAllocator na,
   long index
);
```

Parameters

na

You obtain the note allocator identifier by calling `OpenComponent`.

index

Synthesizer to unregister. The value is 1 through the registered music component count returned by `NAGetRegisteredMusicDevice` (page 1722).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`QuickTimeMusic.h`

NAUnrollNoteChannel

Marks a note channel as available to be stolen. (Deprecated in Mac OS X v10.5.)

```
ComponentResult NAUnrollNoteChannel (
   NoteAllocator na,
   NoteChannel noteChannel
);
```

Parameters

na
You obtain the note allocator identifier by calling `OpenComponent`.

noteChannel
Note channel to be unrolled. You obtain the note channel identifier from `NANewNoteChannel` (page 1724) or `NANewNoteChannelFromAtomicInstrument` (page 1724).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Related Sample Code
QTMusicToo

Declared In
`QuickTimeMusic.h`

NewMusicMIDISendUPP

Allocates a Universal Procedure Pointer for the MusicMIDISendProc callback. (Deprecated in Mac OS X v10.5.)

```
MusicMIDISendUPP NewMusicMIDISendUPP (
   MusicMIDISendProcPtr userRoutine
);
```

Parameters

userRoutine
A pointer to your application-defined function.

Return Value
A new UPP; see `Universal Procedure Pointers`.

Discussion
This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes
Introduced in QuickTime 4.1. Replaces `NewMusicMIDISendProc`.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In
`QuickTimeMusic.h`

NewMusicOfflineDataUPP

Allocates a Universal Procedure Pointer for the MusicOfflineDataProc callback. (Deprecated in Mac OS X v10.5.)

```
MusicOfflineDataUPP NewMusicOfflineDataUPP (
   MusicOfflineDataProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your application-defined function.

Return Value
A new UPP; see `Universal Procedure Pointers`.

Discussion
This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes
Introduced in QuickTime 4.1. Replaces `NewMusicOfflineDataProc`.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In
`QuickTimeMusic.h`

NewTuneCallBackUPP

Allocates a Universal Procedure Pointer for the TuneCallBackProc callback. (Deprecated in Mac OS X v10.5.)

```
TuneCallBackUPP NewTuneCallBackUPP (
   TuneCallBackProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your application-defined function.

Return Value
A new UPP; see `Universal Procedure Pointers`.

Discussion
This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes
Introduced in QuickTime 4.1. Replaces `NewTuneCallBackProc`.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In
`QuickTimeMusic.h`

NewTunePlayCallBackUPP

Allocates a Universal Procedure Pointer for the TunePlayCallBackProc callback. (Deprecated in Mac OS X v10.5.)

```
TunePlayCallBackUPP NewTunePlayCallBackUPP (
   TunePlayCallBackProcPtr userRoutine
);
```

Parameters

`userRoutine`

A pointer to your application-defined function.

Return Value
A new UPP; see `Universal Procedure Pointers`.

Discussion
This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes
Introduced in QuickTime 4.1. Replaces `NewTunePlayCallBackProc`.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In
`QuickTimeMusic.h`

QTMIDIGetMIDIPorts

Returns two lists of MIDI ports supported by the specified MIDI component: a list of ports that can receive MIDI input and a list of ports that can send MIDI output. (Deprecated in Mac OS X v10.5.)

```
ComponentResult QTMIDIGetMIDIPorts (
   QTMIDIComponent ci,
   QTMIDIPortListHandle *inputPorts,
   QTMIDIPortListHandle *outputPorts
);
```

Parameters

ci

A MIDI component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

inputPorts

A list of the MIDI ports supported by the component that can receive MIDI input.

outputPorts

A list of the MIDI ports supported by the component that can send MIDI output.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The caller of this function must dispose of the `inputPorts` and `outputPorts` handles.

Special Considerations

`NAGetMIDIPorts` (page 1720) is the correct call for applications to make. They should not call this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`QuickTimeMusic.h`

QTMIDISendMIDI

Sends MIDI data to a MIDI port. (Deprecated in Mac OS X v10.5.)

```
ComponentResult QTMIDISendMIDI (
   QTMIDIComponent ci,
   long portIndex,
   MusicMIDIPacket *mp
);
```

Parameters

ci

A MIDI component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

portIndex

The index of the MIDI port to use for this operation.

mp

A pointer to the MIDI data packet to send.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function can be called at interrupt time. However, the same interrupt level is used whenever MIDI data is sent by the specified MIDI component.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`QuickTimeMusic.h`

QTMIDIUseSendPort

Allocates a MIDI port for output or to release the port. (Deprecated in Mac OS X v10.5.)

```
ComponentResult QTMIDIUseSendPort (
   QTMIDIComponent ci,
   long portIndex,
   long inUse
);
```

Parameters

ci
: A MIDI component instance. Your software obtains this reference from `OpenComponent` or `OpenDefaultComponent`.

portIndex
: The index of the MIDI port for this operation.

inUse
: Specifies whether to allocate the MIDI port for output (if the value is 1) or to release the port (if the value is 0).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`QuickTimeMusic.h`

TuneGetIndexedNoteChannel

Determines how many parts a tune is playing and which instrument is assigned to those parts. (Deprecated in Mac OS X v10.5.)

```
ComponentResult TuneGetIndexedNoteChannel (
   TunePlayer tp,
   long i,
   NoteChannel *nc
);
```

Parameters

tp

A tune player identifier, obtained from `OpenComponent` or `OpenDefaultComponent`.

i

Note channel index, or 0 to get the number of parts.

nc

A pointer to an allocated initialized note channel.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The tune player allocates note channels that best satisfy the requested instrument in the tune header. The application can use this call to determine which instrument was actually used for each note channel. This function takes the tune player in the `tp` parameter and returns the number of parts (1...n) allocated to the tune player. You can then pass the function a part index and it returns, in the `nc` parameter, the note channel allocated for that part.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Related Sample Code

QTMusicToo

Declared In

`QuickTimeMusic.h`

TuneGetNoteAllocator

Returns the instance of the note allocator that the tune player is using. (Deprecated in Mac OS X v10.5.)

```
NoteAllocator TuneGetNoteAllocator (
   TunePlayer tp
);
```

Parameters

tp

A tune player identifier, obtained from `OpenComponent` or `OpenDefaultComponent`.

Return Value

A note allocator or an error code. See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`QuickTimeMusic.h`

TuneGetPartMix

Gets volume, balance, and mixing settings for a specified part of a tune. (Deprecated in Mac OS X v10.5.)

```
ComponentResult TuneGetPartMix (
   TunePlayer tp,
   unsigned long partNumber,
   long *volumeOut,
   long *balanceOut,
   long *mixFlagsOut
);
```

Parameters

tp

A tune player identifier, obtained from `OpenComponent` or `OpenDefaultComponent`.

partNumber

The part number for this request.

volumeOut

Returns the volume for the part.

balanceOut

Returns the balance for the part.

mixFlagsOut

Returns flags (see below) that control part mixing. See these constants:

`kTuneMixMute`

`kTuneMixSolo`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`QuickTimeMusic.h`

TuneGetStatus

Returns an initialized structure describing the state of the tune player instance. (Deprecated in Mac OS X v10.5.)

```
ComponentResult TuneGetStatus (
   TunePlayer tp,
   TuneStatus *status
);
```

Parameters

tp

A tune player identifier, obtained from `OpenComponent` or `OpenDefaultComponent`.

status

A pointer to an initialized `TuneStatus` structure.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Related Sample Code

qtmusic

qtmusic.win

Declared In

`QuickTimeMusic.h`

TuneGetTimeBase

Returns the time base of the tune player. (Deprecated in Mac OS X v10.5.)

```
ComponentResult TuneGetTimeBase (
   TunePlayer tp,
   TimeBase *tb
);
```

Parameters

tp

A tune player identifier, obtained from `OpenComponent` or `OpenDefaultComponent`.

tb

A pointer to a time base identifier, such as that returned by `NewTimeBase` (page 261). On return, the time base used to control the sequence timing.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The sequence can be controlled in several ways through its time base. The rate of playback can be changed, or the time base object can be slaved to a clock or time base different than real time.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Related Sample Code
QTMusicToo

Declared In
`QuickTimeMusic.h`

TuneGetTimeScale

Returns the current time scale for a specified tune player instance. (Deprecated in Mac OS X v10.5.)

```
ComponentResult TuneGetTimeScale (
   TunePlayer tp,
   TimeScale *scale
);
```

Parameters

tp
A tune player identifier, obtained from `OpenComponent` or `OpenDefaultComponent`.

scale
A pointer to an initialized `TimeScale` variable that indicates the tune player's current time scale in units per second.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Related Sample Code
QTMusicToo

Declared In
`QuickTimeMusic.h`

TuneGetVolume

Returns the volume associated with an entire tune sequence. (Deprecated in Mac OS X v10.5.)

```
ComponentResult TuneGetVolume (
   TunePlayer tp
);
```

Parameters

tp

A tune player identifier, obtained from `OpenComponent` or `OpenDefaultComponent`.

Return Value

The volume as a value from 0.0 to 1.0, or a negative result code. See `Error Codes`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`QuickTimeMusic.h`

TuneInstant

Plays a particular sequence of events active at a specified position. (Deprecated in Mac OS X v10.5.)

```
ComponentResult TuneInstant (
   TunePlayer tp,
   unsigned long *tune,
   unsigned long tunePosition
);
```

Parameters

tp

A tune player identifier, obtained from `OpenComponent` or `OpenDefaultComponent`.

tune

A pointer to tune sequence data.

tunePosition

The position within the tune sequence data in time units.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function plays the notes that are "on" at the point specified by the `tunePosition` parameter. The notes are started and then left playing on return. The notes can be silenced by calling `TuneStop` (page 1759). This call is useful for enabling user "scrubbing" on a sequence.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Related Sample Code
QTMusicToo

Declared In
`QuickTimeMusic.h`

TunePreroll

Prepares to play a tune player sequence data by attempting to reserve note channels for each part in the sequence. (Deprecated in Mac OS X v10.5.)

```
ComponentResult TunePreroll (
   TunePlayer tp
);
```

Parameters

tp

A tune player identifier, obtained from `OpenComponent` or `OpenDefaultComponent`.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Related Sample Code
QTMusicToo

Declared In
`QuickTimeMusic.h`

TuneQueue

Places a sequence of music events into a queue to be played. (Deprecated in Mac OS X v10.5.)

```
ComponentResult TuneQueue (
   TunePlayer tp,
   unsigned long *tune,
   Fixed tuneRate,
   unsigned long tuneStartPosition,
   unsigned long tuneStopPosition,
   unsigned long queueFlags,
   TuneCallBackUPP callBackProc,
   long refCon
);
```

Parameters

tp

A tune player identifier, obtained from `OpenComponent` or `OpenDefaultComponent`.

tune
: A pointer to an array of events, terminated by a marker event of subtype `kMarkerEventEnd`. See `QTMA Events`.

tuneRate
: Speed at which to play the sequence. "Normal" speed is 0x00010000.

tuneStartPosition
: Sequence starting time.

tuneStopPosition
: Sequence stopping time. The `tuneStartPosition` and `tuneStopPosition` parameters specify, in time units numbered from 0 for the beginning of the sequence, which part of the queued sequence to play. To play all of it, pass 0 and 0xFFFFFFFF, respectively.

queueFlags
: Flags (see below) with details about how to play the queued tunes. See these constants:
  - `kTuneStartNow`
  - `kTuneDontClipNotes`
  - `kTuneExcludeEdgeNotes`
  - `kTuneQuickStart`
  - `kTuneLoopUntil`
  - `kTuneStartNewMaster`

callBackProc
: A pointer to a `TuneCallBackProc` callback.

refCon
: A reference constant to be passed to your `TuneCallBackProc` callback. Use this parameter to point to a data structure containing any information your function needs.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Related Sample Code
qtmusic

qtmusic.win

QTMusicToo

Declared In
`QuickTimeMusic.h`

TuneSetBalance

Modifies the pan controller setting for a tune player. (Deprecated in Mac OS X v10.5.)

```
ComponentResult TuneSetBalance (
   TunePlayer tp,
   long balance
);
```

Parameters

tp

A tune player identifier, obtained from `OpenComponent` or `OpenDefaultComponent`.

balance

A new pan controller setting. Valid values are from -128 to 128 for left-to-right balance.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`QuickTimeMusic.h`

TuneSetHeader

Prepares the tune player to accept subsequent music event sequences by defining one or more parts to be used by sequence Note events. (Deprecated in Mac OS X v10.5.)

```
ComponentResult TuneSetHeader (
   TunePlayer tp,
   unsigned long *header
);
```

Parameters

tp

A tune player identifier, obtained from `OpenComponent` or `OpenDefaultComponent`.

header

A pointer to a list of instruments that will be used in subsequent calls to the `TuneQueue` function. The list can include events with subtypes of `kGeneralEventNoteRequest`, `kGeneralEventPartKey`, `kGeneralEventAtomicInstrument`, `kGeneralEventMIDIChannel`, and `kGeneralEventUsedNotes`. It can also include atomic instruments. The list is terminated by a marker event of subtype `kMarkerEventEnd`. See `QTMA Events`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function is the first QuickTime music architecture call to play a music sequence. The header parameter points to one or more initialized General events and atomic instruments. Only one call to this function is required. Each call to this function resets the tune player.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Related Sample Code

qtmusic

qtmusic.win

QTMusicToo

Declared In

`QuickTimeMusic.h`

TuneSetHeaderWithSize

Similar to TuneSetHeader but lets you specify the header length. (Deprecated in Mac OS X v10.5.)

```
ComponentResult TuneSetHeaderWithSize (
   TunePlayer tp,
   unsigned long *header,
   unsigned long size
);
```

Parameters

`tp`

A tune player identifier, obtained from `OpenComponent` or `OpenDefaultComponent`.

`header`

A pointer to a list of instruments that will be used in subsequent calls to the `TuneQueue` function. The list can include events with subtypes of `kGeneralEventNoteRequest`, `kGeneralEventPartKey`, `kGeneralEventAtomicInstrument`, `kGeneralEventMIDIChannel`, and `kGeneralEventUsedNotes`. It can also include atomic instruments. The list is terminated by a marker event of subtype `kMarkerEventEnd`. See `QTMA Events`.

`size`

The size of the header in bytes.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function resembles `TuneSetHeader` (page 1753) in that it prepares the tune player to accept subsequent music event sequences by defining one or more parts to be used by sequence Note events. But unlike `TuneSetHeader`, it allows you to specify the header length in bytes. This prevents the call from parsing off the end if the music event sequence is missing an end marker.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`QuickTimeMusic.h`

TuneSetNoteChannels

Assigns note channels to a tune player. (Deprecated in Mac OS X v10.5.)

```
ComponentResult TuneSetNoteChannels (
   TunePlayer tp,
   unsigned long count,
   NoteChannel *noteChannelList,
   TunePlayCallBackUPP playCallBackProc,
   long refCon
);
```

Parameters

tp

A tune player identifier, obtained from `OpenComponent` or `OpenDefaultComponent`.

count

The number of note channels to assign.

noteChannelList

A pointer to the list of note channels to assign. The parts for the note channels you assign are numbered from 1 to the value of the `count` parameter.

playCallBackProc

A pointer to a `TunePlayCallBackProc` callback that is called for each event whose part number is greater than the value of the `count` parameter. Events whose part numbers are less than or equal to the value of the `count` parameter are passed to the note channel rather than the callback. This lets you to use the tune player as a general purpose timer/sequencer.

refCon

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

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

When you call this function, any note channels that were previously assigned to the tune player are no longer used and are disposed of.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`QuickTimeMusic.h`

TuneSetPartMix

Sets volume, balance, and mixing settings for a specified part of a tune. (Deprecated in Mac OS X v10.5.)

```
ComponentResult TuneSetPartMix (
   TunePlayer tp,
   unsigned long partNumber,
   long volume,
   long balance,
   long mixFlags
);
```

Parameters

tp

A tune player identifier, obtained from `OpenComponent` or `OpenDefaultComponent`.

partNumber

The part number for this request.

volume

The volume for the part.

balance

The balance for the part.

mixFlags

Flags (see below) that control part mixing. See these constants:

`kTuneMixMute`

`kTuneMixSolo`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`QuickTimeMusic.h`

TuneSetPartTranspose

Modifies the pitch and volume of every note of a tune. (Deprecated in Mac OS X v10.5.)

```
ComponentResult TuneSetPartTranspose (
   TunePlayer tp,
   unsigned long part,
   long transpose,
   long velocityShift
);
```

Parameters

tp

A tune player identifier, obtained from `OpenComponent` or `OpenDefaultComponent`.

part

The part for which you want to change pitch and volume.

transpose

A value by which to modify the pitch of the note. The value is a small integer for semitones or an 8.8 fixed-point number for microtones.

velocityShift

A value to add to the `velocity` parameter passed to `NAPlayNote` (page 1729).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`QuickTimeMusic.h`

TuneSetSofter

Adjusts the volume a tune is played at to the softer volume produced by QuickTime 2.1. (Deprecated in Mac OS X v10.5.)

```
ComponentResult TuneSetSofter (
   TunePlayer tp,
   long softer
);
```

Parameters

tp

A tune player identifier, obtained from `OpenComponent` or `OpenDefaultComponent`.

softer

A value of 1 means play at the QuickTime 2.1 volume; a value of 0 means don't make the volume softer.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function adjusts the volume a tune is played at to the softer volume produced by QuickTime 2.1. Files imported with QuickTime 2.1 automatically play softer. Files imported with QuickTime 2.5 or later play at the new, louder volume.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`QuickTimeMusic.h`

TuneSetSoundLocalization

Passes sound localization data to a tune player. (Deprecated in Mac OS X v10.5.)

```
ComponentResult TuneSetSoundLocalization (
   TunePlayer tp,
   Handle data
);
```

Parameters

tp

A tune player identifier, obtained from `OpenComponent` or `OpenDefaultComponent`.

data

The sound localization data to be passed.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`QuickTimeMusic.h`

TuneSetTimeScale

Sets the time scale used by the specified tune player instance. (Deprecated in Mac OS X v10.5.)

```
ComponentResult TuneSetTimeScale (
   TunePlayer tp,
   TimeScale scale
);
```

Parameters

tp

A tune player identifier, obtained from `OpenComponent` or `OpenDefaultComponent`.

scale

The time scale value to be used, in units per second.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function sets the time scale data used by the tune player's sequence data when interpreting time-based events.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Related Sample Code
QTMusicToo

Declared In
`QuickTimeMusic.h`

TuneSetVolume

Sets the volume for an entire sequence. (Deprecated in Mac OS X v10.5.)

```
ComponentResult TuneSetVolume (
   TunePlayer tp,
   Fixed volume
);
```

Parameters

tp

A tune player identifier, obtained from `OpenComponent` or `OpenDefaultComponent`.

volume

The volume to use for the sequence. The value is a fixed 16.16 number.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function sets the volume level of the active sequence to the value of the `volume` parameter, ranging from 0.0 to 1.0. Individual instruments within the sequence can maintain independent volume levels.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In
`QuickTimeMusic.h`

TuneStop

Stops a currently playing sequence. (Deprecated in Mac OS X v10.5.)

```
ComponentResult TuneStop (
   TunePlayer tp,
   long stopFlags
);
```

Parameters

tp

A tune player identifier, obtained from `OpenComponent` or `OpenDefaultComponent`.

stopFlags

Set to 0.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`QuickTimeMusic.h`

TuneTask

Lets a tune player to perform tasks it must perform at foreground task time. (Deprecated in Mac OS X v10.5.)

```
ComponentResult TuneTask (
   TunePlayer tp
);
```

Parameters

tp

A tune player identifier, obtained from `OpenComponent` or `OpenDefaultComponent`.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Call this function periodically to allow a tune player to perform certain operations it can performed only at foreground application task time. Specifically, the QuickTime music synthesizer cannot load instruments from disk at interrupt time. As a result, embedded program changes are not performed until this function is called.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`QuickTimeMusic.h`

TuneUnroll

Releases any note channel resources that may have been locked down by previous calls to TunePreroll for this tune player. (Deprecated in Mac OS X v10.5.)

```
ComponentResult TuneUnroll (
   TunePlayer tp
);
```

Parameters

tp

A tune player identifier, obtained from `OpenComponent` or `OpenDefaultComponent`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Related Sample Code

QTMusicToo

Declared In

`QuickTimeMusic.h`

Callbacks

MusicMIDISendProc

Undocumented

```
typedef ComponentResult (*MusicMIDISendProcPtr) (ComponentInstance self, long
refCon, MusicMIDIPacket *mmp);
```

If you name your function `MyMusicMIDISendProc`, you would declare it this way:

```
ComponentResult MyMusicMIDISendProc (
    ComponentInstance    self,
    long                 refCon,
    MusicMIDIPacket      *mmp );
```

Parameters

self

Undocumented

refCon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

mmp

A pointer to a `MusicMIDIPacket` structure.

Return Value

See `Error Codes`. Your callback should return `noErr` if there is no error.

Declared In
`QuickTimeMusic.h`

MusicOfflineDataProc

Undocumented

```
typedef ComponentResult (*MusicOfflineDataProcPtr) (Ptr SoundData, long numBytes,
 long myRefCon);
```

If you name your function `MyMusicOfflineDataProc`, you would declare it this way:

```
ComponentResult MyMusicOfflineDataProc (
    Ptr     SoundData,
    long    numBytes,
    long    myRefCon );
```

Parameters

SoundData
: *Undocumented*

numBytes
: *Undocumented*

myRefCon
: *Undocumented*

Return Value
See `Error Codes`. Your callback should return `noErr` if there is no error.

Declared In
`QuickTimeMusic.h`

TuneCallBackProc

Called when a sequence of music events is placed into a queue to be played.

```
typedef void (*TuneCallBackProcPtr) (const TuneStatus *status, long refCon);
```

If you name your function `MyTuneCallBackProc`, you would declare it this way:

```
void MyTuneCallBackProc (
    const TuneStatus    *status,
    long                refCon );
```

Parameters

status
: A pointer to a `TuneStatus` structure.

refCon
: A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

Declared In
`QuickTimeMusic.h`

TunePlayCallBackProc

Supports the TuneSetNoteChannels function.

```
typedef void (*TunePlayCallBackProcPtr) (unsigned long *event, long seed, long
refCon);
```

If you name your function `MyTunePlayCallBackProc`, you would declare it this way:

```
void MyTunePlayCallBackProc (
    unsigned long    *event,
    long             seed,
    long             refCon );
```

Parameters

event

A pointer to a QuickTime music event structure in the sequence data.

seed

A 32-bit value that is guaranteed to be different for each call to the callback routine (unless 2^32 calls are made, after which the values repeat), with one exception: the value passed at the beginning of a note is also passed at the end of the note's duration, together with a note structure or an extended note in which the velocity bits are set to 0.

refCon

A reference constant that the client code supplies to the callback.

Declared In

`QuickTimeMusic.h`

Data Types

AtomicInstrument

Represents a type used by the Music Architecture API.

```
typedef Handle AtomicInstrument;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeMusic.h`

AtomicInstrumentPtr

Represents a type used by the Music Architecture API.

```
typedef Ptr AtomicInstrumentPtr;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeMusic.h`

GCPart

Defines a part in the QuickTime Music Architecture.

```
struct GCPart {
     long                 hwInstrumentNumber;
     short                controller[128];
     long                 volume;
     long                 polyphony;
     long                 midiChannel;
     GCInstrumentData     id;
 };
```

Fields

`hwInstrumentNumber`

Discussion
The instrument number of the instrument for the part.

`controller`

Discussion
An array of 128 bits identifying the available controllers; see `Music Controllers`. Bits are numbered from 1 to 128, starting with the most significant bit of the long word and continuing to the least significant of the last bit.

`volume`

Discussion
The sound volume for this part, ranging from -1.0 to +1.0. The high-order 8 bits contain the integer part; the low-order 8 bits contain the fractional part. A value of +1.0 constitutes the maximum volume of the user's computer. Negative values are silent but retain the magnitude of the volume setting.

`polyphony`

Discussion
The maximum number of voices.

`midiChannel`

Discussion
The system MIDI channel or, for a hardware device, the slot number.

`id`

Discussion
A `GCInstrumentData` structure.

Related Functions
`MusicDerivedSetInstrument` (page 1679)
`MusicDerivedSetKnob` (page 1680)
`MusicDerivedSetPart` (page 1682)
`MusicDerivedSetPartInstrumentNumber` (page 1682)
`MusicDerivedStorePartInstrument` (page 1683)
`MusicGenericGetPart` (page 1687)

Declared In
`QuickTimeMusic.h`

GenericKnobDescription

Describes a knob for the generic music component.

```
struct GenericKnobDescription {
     KnobDescription    kd;
     long               hw1;
     long               hw2;
     long               hw3;
     long               settingsID;
 };
```

Fields
`kd`

Discussion
A `KnobDescription` structure.

`hw1`

Discussion
Undocumented

`hw2`

Discussion
Undocumented

`hw3`

Discussion
Undocumented

`settingsID`

Discussion
Undocumented

Discussion
Undocumented

Related Functions
`MusicDerivedSetKnob` (page 1680)

Declared In
`QuickTimeMusic.h`

GenericKnobDescriptionListHandle

Represents a type used by the Music Architecture API.

```
typedef GenericKnobDescriptionListPtr * GenericKnobDescriptionListHandle;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeMusic.h`

GenericKnobDescriptionListPtr

Represents a type used by the Music Architecture API.

```
typedef GenericKnobDescriptionList * GenericKnobDescriptionListPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeMusic.h`

InstrumentAboutInfo

Contains the information that appears in an instrument's About box and is returned by MusicGetInstrumentAboutInfo.

```
struct InstrumentAboutInfo {
     PicHandle    p;
     Str255       author;
     Str255       copyright;
     Str255       other;
 };
```

Fields
`p`
Discussion
A handle to a graphic for the About box.

`author`
Discussion
The author's name.

`copyright`
Discussion
The copyright information.

`other`
Discussion
Any other textual information.

Related Functions
`MusicGetInstrumentAboutInfo` (page 1691)

Declared In
`QuickTimeMusic.h`

InstrumentInfoListHandle

Represents a type used by the Music Architecture API.

```
typedef InstrumentInfoListPtr * InstrumentInfoListHandle;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeMusic.h`

InstrumentInfoListPtr

Represents a type used by the Music Architecture API.

```
typedef InstrumentInfoList * InstrumentInfoListPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeMusic.h`

KnobDescription

Contains sound parameter values for a single knob.

```
struct KnobDescription {
     Str63    name;
     long     lowValue;
     long     highValue;
     long     defaultValue;
     long     flags;
     long     knobID;
 };
```

Fields
`name`

Discussion
The name of the knob.

`lowValue`

Discussion
The lowest number you can set the knob to.

`highValue`

Discussion
The highest number you can set the knob to.

`defaultValue`

Discussion
A value to use for the default. A default instrument is made of all default values.

`flags`

Discussion

Constants (see below) that provide various items of information about the knob. See these constants:

- `kKnobReadOnly`
- `kKnobInterruptUnsafe`
- `kKnobKeyrangeOverride`
- `kKnobGroupStart`
- `kKnobFixedPoint8`
- `kKnobFixedPoint16`
- `kKnobTypeNumber`
- `kKnobTypeGroupName`
- `kKnobTypeBoolean`
- `kKnobTypeNote`
- `kKnobTypePan`
- `kKnobTypeInstrument`
- `kKnobTypeSetting`
- `kKnobTypeMilliseconds`
- `kKnobTypePercentage`
- `kKnobTypeHertz`
- `kKnobTypeButton`

`knobID`

Discussion

A knob ID or index. A nonzero value in the high byte indicates that it is an ID. The knob index ranges from 1 to the number of knobs; the ID is an arbitrary number. Use the knob ID to refer to the knob in preference to the knob index, which may change.

Related Functions

`MusicGetDrumKnobDescription` (page 1689)
`MusicGetInstrumentKnobDescription` (page 1693)
`MusicGetKnobDescription` (page 1695)

Declared In

`QuickTimeMusic.h`

MusicComponent

Represents a type used by the Music Architecture API.

```
typedef ComponentInstance MusicComponent;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeMusic.h`

MusicController

Represents a type used by the Music Architecture API.

```
typedef SInt32 MusicController;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeMusic.h`

MusicMIDIPacket

Describes MIDI data passed by note allocation calls.

```
struct MusicMIDIPacket {
     unsigned short    length;
     unsigned long     reserved;
     UInt8             data[249];
 };
```

Fields
`length`

Discussion
The length of the data in the packet.

`reserved`

Discussion
Contains 0, or one of the music packet status constants (see below). See these constants:

`kMusicPacketPortLost`

`kMusicPacketPortFound`

`kMusicPacketTimeGap`

`data`

Discussion
MIDI data.

Related Functions
`MusicDerivedMIDISend` (page 1678)
`MusicMIDIReadHookProc`
`MusicMIDISendProc`
`MusicSendMIDI` (page 1704)
`NASendMIDI` (page 1732)
`QTMIDISendMIDI` (page 1744)

Declared In
`QuickTimeMusic.h`

MusicMIDISendUPP

Represents a type used by the Music Architecture API.

```
typedef STACK_UPP_TYPE(MusicMIDISendProcPtr) MusicMIDISendUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeMusic.h`

MusicOfflineDataUPP

Represents a type used by the Music Architecture API.

```
typedef STACK_UPP_TYPE(MusicOfflineDataProcPtr) MusicOfflineDataUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeMusic.h`

NoteAllocator

Represents a type used by the Music Architecture API.

```
typedef ComponentInstance NoteAllocator;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeMusic.h`

NoteChannel

Represents a type used by the Music Architecture API.

```
typedef struct OpaqueNoteChannel * NoteChannel;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeMusic.h`

NoteRequest

Provides complete information for allocating a note channel.

```
struct NoteRequest {
     NoteRequestInfo    info;
     ToneDescription    tone;
 };
```

Fields

`info`

Discussion

A `NoteRequestInfo` structure.

`tone`

Discussion

A `ToneDescription` structure.

Related Functions

`NAGetNoteRequest` (page 1722)
`NANewNoteChannel` (page 1724)

Declared In

`QuickTimeMusic.h`

QTMIDIComponent

Represents a type used by the Music Architecture API.

```
typedef ComponentInstance QTMIDIComponent;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeMusic.h`

QTMIDIPortListHandle

Represents a type used by the Music Architecture API.

```
typedef QTMIDIPortListPtr * QTMIDIPortListHandle;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeMusic.h`

QTMIDIPortListPtr

Represents a type used by the Music Architecture API.

```
typedef QTMIDIPortList * QTMIDIPortListPtr;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeMusic.h`

Str31

Represents a type used by the Music Architecture API.

```
typedef unsigned char Str31;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`IOMacOSTypes.h`

SynthesizerConnections

Describes how a MIDI device is connected to the user's computer.

```
struct SynthesizerConnections {
     OSType    clientID;
     OSType    inputPortID;
     OSType    outputPortID;
     long      midiChannel;
     long      flags;
     long      unique;
     long      reserved1;
     long      reserved2;
 };
```

Fields
`clientID`

Discussion
The client ID provided by the MIDI Manager, or `'OMS '` for an OMS port.

`inputPortID`

Discussion
The ID provided by the MIDI Manager or OMS for the port used to SEND to the MIDI synthesizer.

`outputPortID`

Discussion
The ID provided by the MIDI Manager or OMS for the port that RECEIVES from a keyboard or other control device.

`midiChannel`

Discussion
The system MIDI channel or, for a hardware device, the slot number.

`flags`

Discussion

Constants (see below) that provide information about the type of connection. See these constants:

    kSynthesizerConnectionMMgr
    kSynthesizerConnectionOMS
    kSynthesizerConnectionQT
    kSynthesizerConnectionFMS

`unique`

Discussion

A unique ID you can use instead of an index to identify the synthesizer to the note allocator.

`reserved1`

Discussion

Reserved. Set to 0.

`reserved2`

Discussion

Reserved. Set to 0.

Related Functions

`NAGetRegisteredMusicDevice` (page 1722)
`NARegisterMusicDevice` (page 1730)

Declared In

`QuickTimeMusic.h`

SynthesizerDescription

Contains information about a synthesizer.

```
struct SynthesizerDescription {
     OSType            synthesizerType;
     Str31             name;
     unsigned long     flags;
     unsigned long     voiceCount;
     unsigned long     partCount;
     unsigned long     instrumentCount;
     unsigned long     modifiableInstrumentCount;
     unsigned long     channelMask;
     unsigned long     drumPartCount;
     unsigned long     drumCount;
     unsigned long     modifiableDrumCount;
     unsigned long     drumChannelMask;
     unsigned long     outputCount;
     unsigned long     latency;
     unsigned long     controllers[4];
     unsigned long     gmInstruments[4];
     unsigned long     gmDrums[4];
 };
```

Fields

`synthesizerType`

Discussion
The synthesizer type. This is the same as the music component subtype.

`name`

Discussion
Text name of the synthesizer type.

`flags`

Discussion
Constants (see below) that provide information about how the synthesizer works. See these constants:

`kSynthesizerDynamicVoice`

`kSynthesizerUsesMIDIPort`

`kSynthesizerMicrotone`

`kSynthesizerHasSamples`

`kSynthesizerMixedDrums`

`kSynthesizerSoftware`

`kSynthesizerHardware`

`kSynthesizerDynamicChannel`

`kSynthesizerHogsSystemChannel`

`kSynthesizerSlowSetPart`

`kSynthesizerOffline`

`kSynthesizerGM`

`voiceCount`

Discussion
Maximum polyphony.

`partCount`

Discussion
Maximum multi-timbrality (and MIDI channels).

`instrumentCount`

Discussion
The number of built-in ROM instruments. This does not include General MIDI instruments.

`modifiableInstrumentCount`

Discussion
The number of slots available for saving user-modified instruments.

`channelMask`

Discussion
Which channels a MIDI device always uses for instruments. Set to 0xFFFF for all channels.

`drumPartCount`

Discussion
The maximum multi-timbrality of drum parts. For synthesizers where drum kits are separated from instruments.

`drumCount`

Discussion
The number of built-in ROM drum kits. This does not include General MIDI drum kits. For synthesizers where drum kits are separated from instruments.

`modifiableDrumCount`

Discussion
The number of slots available for saving user-modified drum kits. For MIDI synthesizers where drum kits are separated from instruments.

`drumChannelMask`

Discussion
Which channels a MIDI device always uses for drum kits. Set to FFFF for all channels.

`outputCount`

Discussion
The number of audio outputs. This is usually 2.

`latency`

Discussion
The response time in microseconds.

`controllers`

Discussion
An array of 128 bits identifying the available controllers; see `Music Controllers`. Bits are numbered from 1 to 128, starting with the most significant bit of the long word and continuing to the least significant of the last bit.

`gmInstruments`

Discussion
An array of 128 bits giving the available General MIDI instruments.

`gmDrums`

Discussion
An array of 128 bits giving the available General MIDI drum kits.

Related Functions
`MusicGetDescription` (page 1688)

Declared In
`QuickTimeMusic.h`

TuneCallBackUPP

Represents a type used by the Music Architecture API.

```
typedef STACK_UPP_TYPE(TuneCallBackProcPtr) TuneCallBackUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeMusic.h`

TunePlayCallBackUPP

Represents a type used by the Music Architecture API.

```
typedef STACK_UPP_TYPE(TunePlayCallBackProcPtr) TunePlayCallBackUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeMusic.h`

TunePlayer

Represents a type used by the Music Architecture API.

```
typedef ComponentInstance TunePlayer;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeMusic.h`

TuneStatus

Provides information on the currently playing tune.

```
struct TuneStatus {
     unsigned long *   tune;
     unsigned long *   tunePtr;
     TimeValue         time;
     short             queueCount;
     short             queueSpots;
     TimeValue         queueTime;
     long              reserved[3];
 };
```

Fields

`tune`

Discussion
The currently playing tune.

`tunePtr`

Discussion
Current position within the playing tune.

`time`

Discussion
Current tune time.

`queueCount`

Discussion
Number of tunes queued up.

`queueSpots`

Discussion
Number of tunes that can be added to the queue.

`queueTime`

Discussion
Total amount of playing time represented by tunes in the queue. This value can be very inaccurate.

`reserved`

Discussion
Reserved; set to 0.

Related Functions
`TuneGetStatus` (page 1748)

Declared In
`QuickTimeMusic.h`

Constants

Generic Music Constants

Constants that represent generic music types.

```
enum {
  kGenericMusicComponentSubtype = 'gene'
};
enum {
  kGenericMusicDoMIDI           = 1 << 0, /* implement normal MIDI messages for
note, controllers, and program changes 0-127 */
  kGenericMusicBank0            = 1 << 1, /* implement instrument bank changes on
 controller 0 */
  kGenericMusicBank32           = 1 << 2, /* implement instrument bank changes on
 controller 32 */
  kGenericMusicErsatzMIDI       = 1 << 3, /* construct MIDI packets, but send them
 to the derived component */
  kGenericMusicCallKnobs       = 1 << 4, /* call the derived component with special
 knob format call */
  kGenericMusicCallParts       = 1 << 5, /* call the derived component with special
 part format call */
  kGenericMusicCallInstrument   = 1 << 6, /* call MusicDerivedSetInstrument for
MusicSetInstrument calls */
  kGenericMusicCallNumber      = 1 << 7, /* call MusicDerivedSetPartInstrumentNumber
 for MusicSetPartInstrumentNumber calls, & don't send any C0 or bank stuff */
  kGenericMusicCallROMInstrument = 1 << 8, /* call MusicSetInstrument for
MusicSetPartInstrumentNumber for "ROM" instruments, passing params from the ROMi
resource */
  kGenericMusicAllDefaults      = 1 << 9 /* indicates that when a new instrument
is recalled, all knobs are reset to DEFAULT settings. True for GS modules */
};
enum {
  kGenericMusicKnob             = 1,
  kGenericMusicInstrumentKnob   = 2,
  kGenericMusicDrumKnob         = 3,
  kGenericMusicGlobalController = 4
};
enum {
  kGenericMusicMiscLongFirst    = 0,
  kGenericMusicMiscLongVoiceCount = 1,
  kGenericMusicMiscLongPartCount = 2,
  kGenericMusicMiscLongModifiableInstrumentCount = 3,
  kGenericMusicMiscLongChannelMask = 4,
  kGenericMusicMiscLongDrumPartCount = 5,
  kGenericMusicMiscLongModifiableDrumCount = 6,
  kGenericMusicMiscLongDrumChannelMask = 7,
  kGenericMusicMiscLongOutputCount = 8,
  kGenericMusicMiscLongLatency  = 9,
  kGenericMusicMiscLongFlags    = 10,
  kGenericMusicMiscLongFirstGMHW = 11,  /* number to add to locate GM main
instruments */
  kGenericMusicMiscLongFirstGMDrumHW = 12, /* number to add to locate GM drumkits
 */
  kGenericMusicMiscLongFirstUserHW = 13, /* First hw number of user instruments
(presumed sequential) */
  kGenericMusicMiscLongLast     = 14
};
enum {
  kGenericMusicResFirst         = 0,
  kGenericMusicResMiscStringList = 1,   /* STR# 1: synth name, 2:about
author,3:aboutcopyright,4:aboutother */
  kGenericMusicResMiscLongList  = 2,    /* Long various params, see list below */
  kGenericMusicResInstrumentList = 3,   /* NmLs of names and shorts, categories
prefixed by two bullet characters */
  kGenericMusicResDrumList      = 4,    /* NmLs of names and shorts */
  kGenericMusicResInstrumentKnobDescriptionList = 5, /* Knob */
  kGenericMusicResDrumKnobDescriptionList = 6, /* Knob */
  kGenericMusicResKnobDescriptionList = 7, /* Knob */
  kGenericMusicResBitsLongList  = 8,    /* Long back to back bitmaps of controllers,
 gminstruments, and drums */
  kGenericMusicResModifiableInstrumentHW = 9, /* Shrt same as the hw shorts trailing
 the instrument names, a shortlist */
  kGenericMusicResGMTranslation = 10,   /* Long 128 long entries, 1 for each gm
inst, of local instrument numbers 1-n (not hw numbers) */
  kGenericMusicResROMInstrumentData = 11, /* knob lists for ROM instruments, so
the knob values may be known */
  kGenericMusicResAboutPICT     = 12,   /* picture for aboutlist. must be present
 for GetAbout call to work */
  kGenericMusicResLast          = 13
};
enum {
  kMusicGenericRange            = 0x0100,
  kMusicDerivedRange            = 0x0200
};
```

Constants

`kGenericMusicAllDefaults`

Indicates that when a new instrument is recalled, all knobs are reset to DEFAULT settings. True for GS modules.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeMusic.h`.

`kGenericMusicDrumKnob`

Value is 3.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeMusic.h`.

`kGenericMusicMiscLongFirstGMHW`

Number to add to locate GM main instruments.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeMusic.h`.

`kGenericMusicMiscLongFirstGMDrumHW`

Number to add to locate GM drumkits.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeMusic.h`.

`kGenericMusicMiscLongFirstUserHW`

First HW number of user instruments (presumed sequential).

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeMusic.h`.

`kGenericMusicResMiscStringList`

STR# 1: `synth` name, 2:about author,3:aboutcopyright,4:aboutother.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeMusic.h`.

`kGenericMusicResMiscLongList`
Long various `params`, see list below.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeMusic.h`.

`kGenericMusicResInstrumentList`
NmLs of names and shorts, categories prefixed by two bullet characters.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeMusic.h`.

`kGenericMusicResDrumList`
NmLs of names and shorts.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeMusic.h`.

`kGenericMusicResInstrumentKnobDescriptionList`
Knob.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeMusic.h`.

`kGenericMusicResDrumKnobDescriptionList`
Knob.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeMusic.h`.

`kGenericMusicResKnobDescriptionList`
Knob.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeMusic.h`.

`kGenericMusicResBitsLongList`
Long back to back bitmaps of controllers, gminstruments, and drums.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeMusic.h`.

`kGenericMusicResModifiableInstrumentHW`
Short same as the HW shorts trailing the instrument names, a short list.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeMusic.h`.

`kGenericMusicResGMTranslation`
Long 128 long entries, 1 for each gm instrument, of local instrument numbers 1-n (not HW numbers).

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeMusic.h`.

`kGenericMusicResROMInstrumentData`
Knob lists for ROM instruments, so the knob values may be known.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeMusic.h`.

`kGenericMusicResAboutPICT`

  `Picture` for about list. Must be present for `GetAbout` call to work.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeMusic.h`.

Declared In
`QuickTimeMusic.h`

MusicSetPartAtomicInstrument Values

Constants passed to MusicSetPartAtomicInstrument.

```
enum {
  kGetAtomicInstNoExpandedSamples = 1 << 0,
  kGetAtomicInstNoOriginalSamples = 1 << 1,
  kGetAtomicInstNoSamples        = kGetAtomicInstNoExpandedSamples |
kGetAtomicInstNoOriginalSamples,
  kGetAtomicInstNoKnobList       = 1 << 2,
  kGetAtomicInstNoInstrumentInfo = 1 << 3,
  kGetAtomicInstOriginalKnobList = 1 << 4,
  kGetAtomicInstAllKnobs         = 1 << 5 /* return even those that are set to
default*/
};
```

Declared In
`QuickTimeMusic.h`

MusicGetInstrumentInfo Values

Constants passed to MusicGetInstrumentInfo.

```
enum {
  kGetInstrumentInfoNoBuiltIn    = 1 << 0,
  kGetInstrumentInfoMidiUserInst = 1 << 1,
  kGetInstrumentInfoNoIText      = 1 << 2
};
```

Declared In
`QuickTimeMusic.h`

kInstrumentMatchGMNumber

Constants grouped with kInstrumentMatchGMNumber.

```
enum {
  kInstrumentMatchSynthesizerType = 1,
  kInstrumentMatchSynthesizerName = 2,
  kInstrumentMatchName          = 4,
  kInstrumentMatchNumber        = 8,
  kInstrumentMatchGMNumber      = 16,
  kInstrumentMatchGSNumber      = 32
};
```

Declared In
`QuickTimeMusic.h`

kKnobBasic

Constants grouped with kKnobBasic.

```
enum {
  kKnobBasic                    = 8,    /* knob shows up in certain simplified
lists of knobs */
  kKnobReadOnly                = 16,   /* knob value cannot be changed by user or
 with a SetKnob call */
  kKnobInterruptUnsafe          = 32,   /* only alter this knob from foreground
task time (may access toolbox) */
  kKnobKeyrangeOverride         = 64,   /* knob can be overridden within a single
 keyrange (software synth only) */
  kKnobGroupStart               = 128,  /* knob is first in some logical group of
 knobs */
  kKnobFixedPoint8              = 1024,
  kKnobFixedPoint16             = 2048, /* One of these may be used at a time. */
  kKnobTypeNumber               = 0 << 12,
  kKnobTypeGroupName            = 1 << 12, /* "knob" is really a group name for
display purposes */
  kKnobTypeBoolean              = 2 << 12, /* if range is greater than 1, its a
multi-checkbox field */
  kKnobTypeNote               = 3 << 12, /* knob range is equivalent to MIDI keys
 */
  kKnobTypePan                  = 4 << 12, /* range goes left/right (lose this? )
 */
  kKnobTypeInstrument           = 5 << 12, /* knob value = reference to another
instrument number */
  kKnobTypeSetting             = 6 << 12, /* knob value is 1 of n different things
 (eg, fm algorithms) popup menu */
  kKnobTypeMilliseconds         = 7 << 12, /* knob is a millisecond time range */
 kKnobTypePercentage          = 8 << 12, /* knob range is displayed as a Percentage
 */
  kKnobTypeHertz                = 9 << 12, /* knob represents frequency */
  kKnobTypeButton               = 10 << 12 /* momentary trigger push button */
};
```

Constants

`kKnobReadOnly`
: The knob value cannot be changed by the user or with a set knob call.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeMusic.h`.

`kKnobInterruptUnsafe`
: Alter this knob only from foreground task time.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeMusic.h`.

`kKnobKeyrangeOverride`
: The knob can be overridden within a single key range (software synthesizer only).

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeMusic.h`.

`kKnobGroupStart`
: The knob is first in some logical group of knobs.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeMusic.h`.

`kKnobFixedPoint8`
: Interpret knob numbers as fixed-point 8-bit.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeMusic.h`.

`kKnobFixedPoint16`
: Interpret knob numbers as fixed-point 16-bit.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeMusic.h`.

`kKnobTypeNumber`
: The knob value is a numerical value.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeMusic.h`.

`kKnobTypeGroupName`
: The name of the knob is really a group name for display purposes.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeMusic.h`.

`kKnobTypeBoolean`
: The knob is an on/off knob. If the range of the knob (as specified by the low value and high value in the knob description structure) is greater than one, the knob is a multi-checkbox field.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeMusic.h`.

`kKnobTypeNote`
: The knob value range is equivalent to MIDI keys.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeMusic.h`.

`kKnobTypePan`
: The knob value is the pan setting and is within a range (as specified by the low value and high value in the knob description structure) that goes from left to right.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeMusic.h`.

`kKnobTypeInstrument`
: The knob value is a reference to another instrument number.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeMusic.h`.

`kKnobTypeSetting`
: The knob value is one of several different discrete settings; for example, items on a pop-up menu.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeMusic.h`.

`kKnobTypeMilliseconds`
: The knob value is in milliseconds.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeMusic.h`.

`kKnobTypePercentage`
: The knob value is a percentage of the range.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeMusic.h`.

`kKnobTypeHertz`
: The knob value represents frequency.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeMusic.h`.

Declared In
`QuickTimeMusic.h`

MusicMIDIPacket Values

Constants passed to MusicMIDIPacket.

```
enum {
  kMusicPacketPortLost          = 1,    /* received when application loses the
default input port */
  kMusicPacketPortFound         = 2,    /* received when application gets it back
 out from under someone else's claim */
  kMusicPacketTimeGap           = 3     /* data[0] = number of milliseconds to keep
 the MIDI line silent */
};
```

Constants

`kMusicPacketPortLost`
: The application has lost the default input port.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeMusic.h`.

`kMusicPacketPortFound`
: The application has retrieved the input port from the previous owner.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeMusic.h`.

Declared In
`QuickTimeMusic.h`

kPickDontMix

Constants grouped with kPickDontMix.

```
enum {
  kPickDontMix                  = 1,    /* dont mix instruments with drum sounds
*/
  kPickSameSynth                = 2,    /* only allow the same device that went
in, to come out */
  kPickUserInsts               = 4,    /* show user insts in addition to ROM voices
 */
  kPickEditAllowEdit            = 8,    /* lets user switch over to edit mode */
  kPickEditAllowPick            = 16,   /* lets the user switch over to pick mode
 */
  kPickEditSynthGlobal          = 32,   /* edit the global knobs of the synth */
  kPickEditControllers          = 64    /* edit the controllers of the notechannel
 */
};
```

Declared In
`QuickTimeMusic.h`

kSetAtomicInstCallerGuarantees

Constants grouped with kSetAtomicInstCallerGuarantees.

```
enum {
  kSetAtomicInstKeepOriginalInstrument = 1 << 0,
  kSetAtomicInstShareAcrossParts = 1 << 1, /* inst disappears when app goes away*/
  kSetAtomicInstCallerTosses    = 1 << 2, /* the caller isn't keeping a copy around
 (for NASetAtomicInstrument)*/
  kSetAtomicInstCallerGuarantees = 1 << 3, /* the caller guarantees a copy is
around*/
  kSetAtomicInstInterruptSafe   = 1 << 4, /* dont move memory at this time (but
process at next task time)*/
  kSetAtomicInstDontPreprocess  = 1 << 7 /* perform no further preprocessing because
 either 1)you know the instrument is digitally clean, or 2) you got it from a
GetPartAtomic*/
};
```

Declared In
`QuickTimeMusic.h`

kSynthesizerConnectionFMS

Constants grouped with kSynthesizerConnectionFMS.

```
enum {
  kSynthesizerConnectionFMS     = 1,    /* this connection imported from FMS */
  kSynthesizerConnectionMMgr    = 2,    /* this connection imported from the MIDI
 Mgr */
  kSynthesizerConnectionOMS     = 4,    /* this connection imported from OMS */
  kSynthesizerConnectionQT     = 8,    /* this connection is a QuickTime-only port
 */
  kSynthesizerConnectionOSXMIDI = 16,   /* this connection is an OS X CoreMIDI port
 */
                                       /* lowest five bits are mutually exclusive;
 combinations reserved for future use.*/
  kSynthesizerConnectionUnavailable = 256 /* port exists, but cannot be used just
 now */
};
```

Constants

`kSynthesizerConnectionFMS`

This connection is imported from the FreeMIDI system.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeMusic.h`.

`kSynthesizerConnectionMMgr`

This connection is imported from the MIDI Manager.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeMusic.h`.

`kSynthesizerConnectionOMS`

This connection is imported from the Open Music System (OMS).

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeMusic.h`.

`kSynthesizerConnectionQT`

This connection is a QuickTime-only port.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeMusic.h`.

Declared In

`QuickTimeMusic.h`

kSynthesizerDLS

Constants grouped with kSynthesizerDLS.

```
enum {
  kSynthesizerDynamicVoice      = 1 << 0, /* can assign voices on the fly (else,
polyphony is very important */
  kSynthesizerUsesMIDIPort      = 1 << 1, /* must be patched through MIDI Manager
 */
  kSynthesizerMicrotone         = 1 << 2, /* can play microtonal scales */
  kSynthesizerHasSamples        = 1 << 3, /* synthesizer has some use for sampled
 data */
  kSynthesizerMixedDrums        = 1 << 4, /* any part can play drum parts, total
= instrument parts */
  kSynthesizerSoftware          = 1 << 5, /* implemented in main CPU software ==
uses cpu cycles */
  kSynthesizerHardware          = 1 << 6, /* is a hardware device (such as nubus,
 or maybe DSP?) */
  kSynthesizerDynamicChannel    = 1 << 7, /* can move any part to any channel or
disable each part. (else we assume it lives on all channels in masks) */
  kSynthesizerHogsSystemChannel = 1 << 8, /* can be channelwise dynamic, but always
 responds on its system channel */
  kSynthesizerHasSystemChannel  = 1 << 9, /* has some "system channel" notion to
distinguish it from multiple instances of the same device (GM devices dont) */
  kSynthesizerSlowSetPart      = 1 << 10, /* SetPart() and SetPartInstrumentNumber()
 calls do not have rapid response, may glitch notes */
  kSynthesizerOffline           = 1 << 12, /* can enter an offline synthesis mode
 */
  kSynthesizerGM                = 1 << 14, /* synth is a GM device */
  kSynthesizerDLS               = 1 << 15, /* synth supports DLS level 1 */
  kSynthesizerSoundLocalization = 1 << 16 /* synth supports extremely baroque,
nonstandard, and proprietary "apple game sprockets" localization parameter set */
};
```

Constants

`kSynthesizerDynamicVoice`

Voices can be assigned to parts on the fly with this synthesizer (otherwise, polyphony is very important).

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeMusic.h`.

`kSynthesizerUsesMIDIPort`

This synthesizer must be patched through a MIDI system, such as the MIDI Manager or OMS.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeMusic.h`.

`kSynthesizerMicrotone`

This synthesizer can play microtonal scales.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeMusic.h`.

`kSynthesizerHasSamples`

This synthesizer has some use for sampled audio data.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeMusic.h`.

`kSynthesizerMixedDrums`

Any part of this synthesizer can play drum parts.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeMusic.h`.

`kSynthesizerSoftware`
: This synthesizer is implemented in main CPU software and uses CPU cycles.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeMusic.h`.

`kSynthesizerHardware`
: This synthesizer is a hardware device, not a software synthesizer or MIDI device.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeMusic.h`.

`kSynthesizerDynamicChannel`
: This synthesizer can move any part to any channel or disable each part. For devices only.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeMusic.h`.

`kSynthesizerHogsSystemChannel`
: Even if the `kSynthesizerDynamicChannel` bit is set, this synthesizer always responds on its system channel. For MIDI devices only.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeMusic.h`.

`kSynthesizerSlowSetPart`
: This synthesizer does not respond rapidly to the various set part and set part instrument calls.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeMusic.h`.

`kSynthesizerOffline`
: This synthesizer can enter an offline synthesis mode.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeMusic.h`.

`kSynthesizerGM`
: This synthesizer is a General MIDI device.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeMusic.h`.

Declared In
`QuickTimeMusic.h`

TuneSetPartMix Values

Constants passed to TuneSetPartMix.

```
enum {
  kTuneMixMute                  = 1,    /* disable a part */
  kTuneMixSolo                  = 2     /* if any parts soloed, play only soloed
parts */
};
```

Declared In
`QuickTimeMusic.h`

kTuneDontClipNotes

Constants grouped with kTuneDontClipNotes.

```
enum {
  kTuneStartNow                = 1,    /* start after buffer is implied */
  kTuneDontClipNotes           = 2,    /* allow notes to finish their durations
outside sample */
  kTuneExcludeEdgeNotes        = 4,    /* dont play notes that start at end of
tune */
  kTuneQuickStart              = 8,    /* Leave all the controllers where they
are, ignore start time */
  kTuneLoopUntil               = 16,   /* loop a queued tune if there's nothing
else in the queue*/
  kTunePlayDifference          = 32,   /* by default, the tune difference is
skipped*/
  kTunePlayConcurrent          = 64,   /* dont block the next tune sequence with
 this one*/
  kTuneStartNewMaster          = 16384
};
```

Declared In
`QuickTimeMusic.h`

QuickTime Streaming Reference

Framework: Frameworks/QuickTime.framework

Declared in QuickTimeStreaming.h, QTStreamingComponents.h

Overview

The streaming API in QuickTime allows developers to recognize and play streaming movies, add hint tracks so movies can be streamed, create packetizers and reassemblers, mix streaming and nonstreaming data in a single movie, and broadcast live streams in real time.

Functions

DisposeQTSModalFilterUPP

Disposes of a QTSModalFilterUPP pointer.

```
void DisposeQTSModalFilterUPP (
   QTSModalFilterUPP userUPP
);
```

Parameters

userUPP

A `QTSModalFilterUPP` pointer.

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeStreaming.h`

DisposeQTSNotificationUPP

Disposes of a QTSNotificationUPP pointer.

```
void DisposeQTSNotificationUPP (
   QTSNotificationUPP userUPP
);
```

Parameters

userUPP

A `QTSNotificationUPP` pointer. See `Universal Procedure Pointers`.

Return Value

You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes

Introduced in QuickTime 4.1.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeStreaming.h`

DisposeQTSPanelFilterUPP

Disposes of a QTSPanelFilterUPP pointer.

```
void DisposeQTSPanelFilterUPP (
   QTSPanelFilterUPP userUPP
);
```

Parameters

userUPP

A `QTSPanelFilterUPP` pointer.

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.1 and later.

Declared In

`QuickTimeStreaming.h`

DisposeRTPMPDataReleaseUPP

Disposes of an RTPMPDataReleaseUPP pointer.

```
void DisposeRTPMPDataReleaseUPP (
   RTPMPDataReleaseUPP userUPP
);
```

Parameters

userUPP

An `RTPMPDataReleaseUPP` pointer. See `Universal Procedure Pointers`.

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

DisposeRTPPBCallbackUPP

Disposes of an RTPPBCallbackUPP pointer.

```
void DisposeRTPPBCallbackUPP (
   RTPPBCallbackUPP userUPP
);
```

Parameters

`userUPP`
An `RTPPBCallbackUPP` pointer. See `Universal Procedure Pointers`.

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

InitializeQTS

Initializes QuickTime streaming.

```
OSErr InitializeQTS (
   void
);
```

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

NewQTSModalFilterUPP

Allocates a Universal Procedure Pointer for the QTSModalFilterProc callback.

```
QTSModalFilterUPP NewQTSModalFilterUPP (
   QTSModalFilterProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your application-defined function.

Return Value
A new UPP; see `Universal Procedure Pointers`.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

NewQTSNotificationUPP

Allocates a Universal Procedure Pointer for the QTSNotificationProc callback.

```
QTSNotificationUPP NewQTSNotificationUPP (
   QTSNotificationProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your application-defined function.

Return Value
A new UPP; see `Universal Procedure Pointers`.

Discussion
This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes
Introduced in QuickTime 4.1. Replaces `NewQTSNotificationProc`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

NewQTSPanelFilterUPP

Allocates a Universal Procedure Pointer for the QTSPanelFilterProc callback.

```
QTSPanelFilterUPP NewQTSPanelFilterUPP (
   QTSPanelFilterProcPtr userRoutine
);
```

Parameters

userRoutine
A pointer to your application-defined function.

Return Value
A new UPP; see `Universal Procedure Pointers`.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.1 and later.

Declared In
`QuickTimeStreaming.h`

NewRTPMPDataReleaseUPP

Allocates a Universal Procedure Pointer for the RTPMPDataReleaseProc callback.

```
RTPMPDataReleaseUPP NewRTPMPDataReleaseUPP (
   RTPMPDataReleaseProcPtr userRoutine
);
```

Parameters

userRoutine
A pointer to your application-defined function.

Return Value
A new UPP; see `Universal Procedure Pointers`.

Discussion
This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes
Introduced in QuickTime 4.1. Replaces `NewRTPMPDataReleaseProc`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

NewRTPPBCallbackUPP

Allocates a Universal Procedure Pointer for the RTPPBCallbackProc callback.

```
RTPPBCallbackUPP NewRTPPBCallbackUPP (
   RTPPBCallbackProcPtr userRoutine
);
```

Parameters

userRoutine
A pointer to your application-defined function.

Return Value
A new UPP; see `Universal Procedure Pointers`.

Discussion
This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes
Introduced in QuickTime 4.1. Replaces `NewRTPPBCallbackProc`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

QTSAllocBuffer

Allocates a QuickTime streaming stream buffer.

```
QTSStreamBuffer * QTSAllocBuffer (
   SInt32 inSize
);
```

Parameters

inSize
The size of the buffer to be allocated.

Return Value
A `QTSStreamBuffer` structure

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSAllocMemPtr

Undocumented

```
QTSMemPtr QTSAllocMemPtr (
   UInt32 inByteCount,
   SInt32 inFlags
);
```

Parameters

inByteCount
    Undocumented

inFlags
    Undocumented

Return Value
Undocumented

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSCopyMessage

Undocumented

```
QTSStreamBuffer * QTSCopyMessage (
   QTSStreamBuffer *inMessage
);
```

Parameters

inMessage
    A pointer to a `QTSStreamBuffer` structure.

Return Value
A pointer to a `QTSStreamBuffer` structure.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSDisposePresentation

Disposes of a QuickTime streaming presentation.

```
OSErr QTSDisposePresentation (
   QTSPresentation inPresentation,
   SInt32 inFlags
);
```

Parameters

inPresentation

A pointer to a `QTSPresentationRecord` structure that defines the presentation to be disposed.

inFlags

Flags governing the disposal of the presentation. Currently, no flags are defined; set this parameter to 0.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeStreaming.h`

QTSDisposeStatHelper

Disposes of a QuickTime streaming statistics helper that was previously created by QTSNewStatHelper.

```
OSErr QTSDisposeStatHelper (
   QTSStatHelper inStatHelper
);
```

Parameters

inStatHelper

A pointer to a `QTSStatHelperRecord` structure that defines the statistics helper to be disposed.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeStreaming.h`

QTSDisposeStream

Disposes of a QuickTime streaming stream.

```
OSErr QTSDisposeStream (
   QTSStream inStream,
   SInt32 inFlags
);
```

Parameters

inStream

A pointer to a `QTSStreamRecord` structure that defines a stream to be disposed.

inFlags

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeStreaming.h`

QTSDuplicateMessage

Undocumented

```
OSErr QTSDuplicateMessage (
   QTSStreamBuffer *inMessage,
   SInt32 inFlags,
   QTSStreamBuffer **outDuplicatedMessage
);
```

Parameters

inMessage

Undocumented

inFlags

Undocumented

outDuplicatedMessage

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeStreaming.h`

QTSDupMessage

Undocumented

```
QTSStreamBuffer * QTSDupMessage (
   QTSStreamBuffer *inMessage
);
```

Parameters

inMessage

A pointer to a `QTSStreamBuffer` structure.

Return Value

A pointer to a `QTSStreamBuffer` structure.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeStreaming.h`

QTSFindMediaPacketizer

Creates a list of media packetizers that can work with a specified sample description and meet specified criteria.

```
OSErr QTSFindMediaPacketizer (
   MediaPacketizerRequirementsPtr inPacketizerinfo,
   SampleDescriptionHandle inSampleDescription,
   RTPPayloadSortRequestPtr inSortInfo,
   QTAtomContainer *outPacketizerList
);
```

Parameters

inPacketizerinfo

A pointer to a `MediaPacketizerRequirements` structure that specifies the required features of the media packetizers you are looking for.

inSampleDescription

A handle to a `SampleDescription` structure that specifies the media data the packetizer needs to work with.

inSortInfo

A pointer to a `RTPPayloadSortRequest` structure that specifies the sort order for the list of packetizers.

outPacketizerList

On entry, a pointer to a handle to a QT atom container. On return, this container will be filled with a sorted list of available media packetizers that meet the specified criteria. Only packetizers that have the features specified by `inPacketizerInfo` will be listed. The list will be sorted in the order specified by `inSortInfo`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

QTSFindMediaPacketizerForPayloadID

Creates a list of media packetizers for a specified payload number.

```
OSErr QTSFindMediaPacketizerForPayloadID (
   long payloadID,
   RTPPayloadSortRequestPtr inSortInfo,
   QTAtomContainer *outPacketizerList
);
```

Parameters

payloadID
An IETF payload number.

inSortInfo
A pointer to a `RTPPayloadSortRequest` structure that specifies the sort order for the list of packetizers.

outPacketizerList
On entry, a pointer to a handle to a QT atom container. On return, this container will be filled with a sorted list of available media packetizers for the specified payload ID. The list will be sorted in the order specified by `inSortInfo`.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

QTSFindMediaPacketizerForPayloadName

Creates a list of media packetizers for a specified payload name.

```
OSErr QTSFindMediaPacketizerForPayloadName (
   const char *payloadName,
   RTPPayloadSortRequestPtr inSortInfo,
   QTAtomContainer *outPacketizerList
);
```

Parameters

payloadName
: A pointer to a payload name string.

inSortInfo
: A pointer to a RTPPayloadSortRequest structure that specifies the sort order for the list of packetizers.

outPacketizerList
: On entry, a pointer to a handle to a QT atom container. On return, this container will be filled with a sorted list of available media packetizers for the specified payload name. The list will be sorted in the order specified by inSortInfo.

Return Value

See Error Codes. Returns noErr if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

QTStreamingComponents.h

QTSFindMediaPacketizerForTrack

Creates a list of media packetizers for a specified movie track and sample data.

```
OSErr QTSFindMediaPacketizerForTrack (
   Track inTrack,
   long inSampleDescriptionIndex,
   RTPPayloadSortRequestPtr inSortInfo,
   QTAtomContainer *outPacketizerList
);
```

Parameters

inTrack
: The track for this operation. Your application obtains this track identifier from such functions as NewMovieTrack (page 1628) and GetMovieTrack (page 1601).

inSampleDescriptionIndex
: The value of the dataRefIndex field of the SampleDescription structure that specifies the type of media data that will be packetized.

inSortInfo
: A pointer to a RTPPayloadSortRequest structure that specifies the sort order for the list of packetizers.

outPacketizerList

On entry, a pointer to a handle to a QT atom container. On return, this container will be filled with a sorted list of available media packetizers for the specified track. The list will be sorted in the order specified by `inSortInfo`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QTStreamingComponents.h`

QTSFindReassemblerForPayloadID

Creates a list of streaming reassemblers for a specified payload number.

```
OSErr QTSFindReassemblerForPayloadID (
   UInt8 inPayloadID,
   RTPPayloadSortRequest *inSortInfo,
   QTAtomContainer *outReassemblerList
);
```

Parameters

inPayloadID

An IETF payload number.

inSortInfo

A pointer to a `RTPPayloadSortRequest` structure that specifies the sort order for the list of reassemblers.

outReassemblerList

On entry, a pointer to a handle to a QT atom container. On return, this container will be filled with a sorted list of available reassemblers for the specified track. The list will be sorted in the order specified by `inSortInfo`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QTStreamingComponents.h`

QTSFindReassemblerForPayloadName

Creates a list of streaming reassemblers for a specified payload name.

```
OSErr QTSFindReassemblerForPayloadName (
   const char *inPayloadName,
   RTPPayloadSortRequest *inSortInfo,
   QTAtomContainer *outReassemblerList
);
```

Parameters

inPayloadName

A payload name string.

inSortInfo

A pointer to a `RTPPayloadSortRequest` structure that specifies the sort order for the list of reassemblers.

outReassemblerList

On entry, a pointer to a handle to a QT atom container. On return, this container will be filled with a sorted list of available reassemblers for the specified track. The list will be sorted in the order specified by `inSortInfo`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QTStreamingComponents.h`

QTSFlattenMessage

Undocumented

```
QTSStreamBuffer * QTSFlattenMessage (
   QTSStreamBuffer *inMessage
);
```

Parameters

inMessage

A pointer to a `QTSStreamBuffer` structure.

Return Value

A pointer to a `QTSStreamBuffer` structure.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeStreaming.h`

QTSFreeMessage

Undocumented

```
void QTSFreeMessage (
   QTSStreamBuffer *inMessage
);
```

Parameters

inMessage

A pointer to a `QTSStreamBuffer` structure.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeStreaming.h`

QTSGetErrorString

Undocumented

```
Boolean QTSGetErrorString (
   SInt32 inErrorCode,
   UInt32 inMaxErrorStringLength,
   char *outErrorString,
   SInt32 inFlags
);
```

Parameters

inErrorCode

Undocumented

inMaxErrorStringLength

Undocumented

outErrorString

Undocumented

inFlags

Undocumented

Return Value

Undocumented

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeStreaming.h`

QTSGetNetworkAppName

Gets the name of a streaming network application.

```
OSErr QTSGetNetworkAppName (
   SInt32 inFlags,
   char **outCStringPtr
);
```

Parameters

inFlags

A flag (see below) that determines whether the application name is a full pathname. See these constants:

`kQTSNetworkAppNameIsFullNameFlag`

outCStringPtr

A `Ptr` to a `CStringPtr`; see `MacTypes.h`. This information is sent back to servers in HTTP and RTSP headers, so they can work out client statistics. A typical default string is `QTS (qtver=4.1.1;cpu=PPC;os=Mac 9.0.4)`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Following is an example of calling this function:

```
Ptr networkAppName =NIL;
err =QTSGetNetworkAppName(0L, &networkAppName);
printf("The NetworkAppName is %s", networkAppName);
DisposePtr(networkAppName);
// This call prints
// The NetworkAppName is QTS (qtver=4.1.1;cpu=PPC;os=Mac 9.0.4)
// or
// The NetworkAppName is QTS (qtver=4.0;os=Windows NT 4.0 Service Pack 3)
// If you set it from your app, that will be returned instead.
```

Version Notes

Introduced in QuickTime 4.1.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeStreaming.h`

QTSGetOrMakeStatAtomForStream

Gets the statistics atom for a stream or creates a new statistics atom for it.

```
OSErr QTSGetOrMakeStatAtomForStream (
   QTAtomContainer inContainer,
   QTSStream inStream,
   QTAtom *outParentAtom
);
```

Parameters

inContainer
An atom container that holds the statistics atoms for the specified stream.

inStream
A pointer to a `QTSStreamRecord` structure that defines a stream.

outParentAtom
On entry, a pointer to a variable of type `QTAtom`; on return, this variable is set to the atom that holds the statistics for this stream. If no such atom exists for that stream, then the function creates a statistics atom.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Special Considerations

This function is to be used only by stream components to put stream statistics into an atom container; applications should not call it.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSGetStreamPresentation

Gets the presentation for a stream.

```
QTSPresentation QTSGetStreamPresentation (
   QTSStream inStream
);
```

Parameters

inStream
A pointer to a `QTSStreamRecord` structure that defines a stream.

Return Value
A pointer to a `QTSPresentationRecord` structure.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSInitializeMediaParams

Undocumented

```
OSErr QTSInitializeMediaParams (
   QTSMediaParams *inMediaParams
);
```

Parameters

inMediaParams

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.1 and later.

Declared In

`QuickTimeStreaming.h`

QTSInsertStatistic

Inserts statistics data into the statistic atom for a stream.

```
OSErr QTSInsertStatistic (
   QTAtomContainer inContainer,
   QTAtom inParentAtom,
   OSType inStatType,
   void *inStatData,
   UInt32 inStatDataLength,
   OSType inStatDataFormat,
   SInt32 inFlags
);
```

Parameters

inContainer

A handle to the atom container that contains the statistic atom.

inParentAtom

The atom that will hold a new atom containing the specified statistic data.

inStatType

A constant (see below) that identifies the type of statistic atom to insert the data into. See these constants:

`kQTSStatisticsStreamAtomType`

`kQTSStatisticsNameAtomType`

`kQTSStatisticsDataFormatAtomType`

`kQTSStatisticsDataAtomType`

`kQTSStatisticsUnitsAtomType`

`kQTSStatisticsUnitsNameAtomType`

inStatData
A pointer to a structure containing the data to insert.

inStatDataLength
The length, in bytes, of the statistic data.

inStatDataFormat
A constant (see below) that identifies the format of the inserted statistic atom. See these constants:
`kQTSStatisticsSInt32DataFormat`
`kQTSStatisticsUInt32DataFormat`
`kQTSStatisticsSInt16DataFormat`
`kQTSStatisticsUInt16DataFormat`
`kQTSStatisticsFixedDataFormat`
`kQTSStatisticsStringDataFormat`
`kQTSStatisticsOSTypeDataFormat`

inFlags
Currently no flags are defined; pass 0 in this parameter.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Special Considerations

This function is to be used only by stream components to put stream statistics into an atom container; applications should not call it.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSInsertStatisticName

Inserts the name and type of a statistic datum into the statistic atom for a stream.

```
OSErr QTSInsertStatisticName (
   QTAtomContainer inContainer,
   QTAtom inParentAtom,
   OSType inStatType,
   const char *inStatName,
   UInt32 inStatNameLength
);
```

Parameters

inContainer
A handle to the atom container that contains the statistic atom. Both the atom container and the parent atom must already exist.

inParentAtom
The atom that will hold a new atom containing the specified statistic name and type.

inStatType

A constant (see below) that identifies the type of statistic atom to insert the data into. See these constants:

`kQTSStatisticsStreamAtomType`

`kQTSStatisticsNameAtomType`

`kQTSStatisticsDataFormatAtomType`

`kQTSStatisticsDataAtomType`

`kQTSStatisticsUnitsAtomType`

`kQTSStatisticsUnitsNameAtomType`

inStatName

A pointer to the name string to be inserted.

inStatNameLength

The length of the name string in characters.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Special Considerations

This function is to be used only by stream components to put stream statistics into an atom container; applications should not call it.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeStreaming.h`

QTSInsertStatisticUnits

Inserts the name and type of statistic units into the statistic atom for a stream.

```
OSErr QTSInsertStatisticUnits (
   QTAtomContainer inContainer,
   QTAtom inParentAtom,
   OSType inStatType,
   OSType inUnitsType,
   const char *inUnitsName,
   UInt32 inUnitsNameLength
);
```

Parameters

inContainer

A handle to the atom container that contains the statistic atom. Both the atom container and the parent atom must already exist.

inParentAtom

The atom that will hold a new atom containing the specified statistic name and type.

inStatType

A constant (see below) that identifies the type of statistic atom to insert the data into. See these constants:

`kQTSStatisticsStreamAtomType`

`kQTSStatisticsNameAtomType`

`kQTSStatisticsDataFormatAtomType`

`kQTSStatisticsDataAtomType`

`kQTSStatisticsUnitsAtomType`

`kQTSStatisticsUnitsNameAtomType`

inUnitsType

A constant (see below) that identifies the type of units atom to insert the data into. See these constants:

`kQTSStatisticsNoUnitsType`

`kQTSStatisticsPercentUnitsType`

`kQTSStatisticsBitsPerSecUnitsType`

`kQTSStatisticsFramesPerSecUnitsType`

inUnitsName

A pointer to the units name string to be inserted.

inUnitsNameLength

The length of the units name string in characters.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Special Considerations

This function is to be used only by stream components to put stream statistics into an atom container; applications should not call it.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeStreaming.h`

QTSMediaGetIndStreamInfo

Undocumented

```
ComponentResult QTSMediaGetIndStreamInfo (
   MediaHandler mh,
   SInt32 inIndex,
   OSType inSelector,
   void *ioParams
);
```

Parameters

mh

A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

inIndex

Undocumented

inSelector

A constant (see below) that identifies the type of information to be retrieved. See these constants:

`kQTSMediaPresentationInfo`

`kQTSMediaNotificationInfo`

`kQTSMediaTotalDataRateInfo`

`kQTSMediaLostPercentInfo`

`kQTSMediaNumStreamsInfo`

`kQTSMediaIndSampleDescriptionInfo`

ioParams

A pointer to returned information in a format determined by `inSelector` (see below).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QTSMovie.h`

QTSMediaGetInfo

Gets information about a streaming media.

```
ComponentResult QTSMediaGetInfo (
   MediaHandler mh,
   OSType inSelector,
   void *ioParams
);
```

Parameters

mh

A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

inSelector

A constant (see below) that identifies the type of information to be retrieved. See these constants:

`kQTSMediaPresentationInfo`

`kQTSMediaNotificationInfo`

`kQTSMediaTotalDataRateInfo`

`kQTSMediaLostPercentInfo`

`kQTSMediaNumStreamsInfo`

`kQTSMediaIndSampleDescriptionInfo`

ioParams

A pointer to returned information in a format determined by `inSelector` (see below).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTSMovie.h`

QTSMediaSetIndStreamInfo

Undocumented

```
ComponentResult QTSMediaSetIndStreamInfo (
   MediaHandler mh,
   SInt32 inIndex,
   OSType inSelector,
   void *ioParams
);
```

Parameters

mh
A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

inIndex
Undocumented

inSelector
A constant (see below) that identifies the type of information to be set. See these constants:

- `kQTSMediaPresentationInfo`
- `kQTSMediaNotificationInfo`
- `kQTSMediaTotalDataRateInfo`
- `kQTSMediaLostPercentInfo`
- `kQTSMediaNumStreamsInfo`
- `kQTSMediaIndSampleDescriptionInfo`

ioParams
A pointer to information in a format determined by `inSelector` (see below).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTSMovie.h`

QTSMediaSetInfo

Sets information about a streaming media.

```
ComponentResult QTSMediaSetInfo (
   MediaHandler mh,
   OSType inSelector,
   void *ioParams
);
```

Parameters

mh

A media handler. You can obtain this reference from `GetMediaHandler` (page 1577).

inSelector

A constant (see below) that identifies the type of information to be set. See these constants:

`kQTSMediaPresentationInfo`

`kQTSMediaNotificationInfo`

`kQTSMediaTotalDataRateInfo`

`kQTSMediaLostPercentInfo`

`kQTSMediaNumStreamsInfo`

`kQTSMediaIndSampleDescriptionInfo`

ioParams

A pointer to information in a format determined by `inSelector` (see below).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QTSMovie.h`

QTSMessageLength

Undocumented

```
UInt32 QTSMessageLength (
   QTSStreamBuffer *inMessage
);
```

Parameters

inMessage

A pointer to a `QTSStreamBuffer` structure.

Return Value

The message length.

Version Notes

Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSNewHandle

Allocates a new handle for data, with options and checking.

```
Handle QTSNewHandle (
   UInt32 inByteCount,
   SInt32 inFlags,
   SInt32 *outFlags
);
```

Parameters

inByteCount
The requested size in bytes of the relocatable block.

inFlags
Flags (see below) that control memory allocation options. See these constants:
- `kQTSMemAllocClearMem`
- `kQTSMemAllocDontUseTempMem`
- `kQTSMemAllocTryTempMemFirst`
- `kQTSMemAllocDontUseSystemMem`
- `kQTSMemAllocTrySystemMemFirst`
- `kQTSMemAllocHoldMemory`
- `kQTSMemAllocIsInterruptTime`

outFlags
A pointer to memory where return flags (see below) report on the block's actual memory location. See these constants:
- `kQTSMemAllocAllocatedInTempMem`
- `kQTSMemAllocAllocatedInSystemMem`

Return Value
The new handle.

Discussion
This function is a handy way to allocate memory without overflowing the application heap, which is mostly a concern with Mac OS versions 7 through 9. It is often used for streaming data.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSNewPresentation

Creates a new streaming presentation.

```
OSErr QTSNewPresentation (
   const QTSNewPresentationParams *inParams,
   QTSPresentation *outPresentation
);
```

Parameters

inParams

A pointer to a `QTSNewPresentationParams` structure that specifies the presentation.

outPresentation

A pointer to a pointer to a new `QTSPresentationRecord` structure.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeStreaming.h`

QTSNewPresentationFromData

Undocumented

```
OSErr QTSNewPresentationFromData (
   OSType inDataType,
   const void *inData,
   const SInt64 *inDataLength,
   const QTSPresParams *inPresParams,
   QTSPresentation *outPresentation
);
```

Parameters

inDataType

Undocumented

inData

Undocumented

inDataLength

Undocumented

inPresParams

Undocumented

outPresentation

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSNewPresentationFromDataRef

Undocumented

```
OSErr QTSNewPresentationFromDataRef (
   Handle inDataRef,
   OSType inDataRefType,
   const QTSPresParams *inPresParams,
   QTSPresentation *outPresentation
);
```

Parameters

inDataRef
    Undocumented

inDataRefType
    Undocumented

inPresParams
    Undocumented

outPresentation
    Undocumented

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSNewPresentationFromFile

Undocumented

```
OSErr QTSNewPresentationFromFile (
   const FSSpec *inFileSpec,
   const QTSPresParams *inPresParams,
   QTSPresentation *outPresentation
);
```

Parameters

inFileSpec
: *Undocumented*

inPresParams
: *Undocumented*

outPresentation
: *Undocumented*

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSNewPtr

Allocates a block of memory for streaming data, with options and checking, and returns a pointer to it.

```
Ptr QTSNewPtr (
   UInt32 inByteCount,
   SInt32 inFlags,
   SInt32 *outFlags
);
```

Parameters

inByteCount
: The requested size in bytes of the new memory block.

inFlags
: Flags (see below) that control memory allocation options. See these constants:

    `kQTSMemAllocClearMem`

    `kQTSMemAllocDontUseTempMem`

    `kQTSMemAllocTryTempMemFirst`

    `kQTSMemAllocDontUseSystemMem`

    `kQTSMemAllocTrySystemMemFirst`

    `kQTSMemAllocHoldMemory`

    `kQTSMemAllocIsInterruptTime`

outFlags
: A pointer to memory where return flags (see below) report on the block's actual memory location. See these constants:
  `kQTSMemAllocAllocatedInTempMem`
  `kQTSMemAllocAllocatedInSystemMem`

Return Value
A pointer to the newly allocated block.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSNewSourcer

Undocumented

```
OSErr QTSNewSourcer (
   void *params,
   const QTSSourcerInitParams *inInitParams,
   SInt32 inFlags,
   ComponentInstance *outSourcer
);
```

Parameters

params
: *Undocumented*

inInitParams
: *Undocumented*

inFlags
: *Undocumented*

outSourcer
: *Undocumented*

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

QTSNewStatHelper

Creates a new statistics helper for a stream or presentation.

```
OSErr QTSNewStatHelper (
   QTSPresentation inPresentation,
   QTSStream inStream,
   OSType inStatType,
   SInt32 inFlags,
   QTSStatHelper *outStatHelper
);
```

Parameters

inPresentation
A pointer to a `QTSPresentationRecord` structure that defines the presentation to keep statistics on. To create a statistics helper for a particular stream, pass in `kQTSInvalidPresentation`.

inStream
A pointer to a `QTSStreamRecord` structure that defines the stream to keep statistics on. To create a statistics helper for a whole presentation, pass in `kQTSAllStreams`.

inStatType
A constant (see below) that defines the type of statistic you want the statistics helper to gather. See these constants:

`kQTSAllStatisticsType`

`kQTSShortStatisticsType`

`kQTSSummaryStatisticsType`

inFlags
Constants (see below) governing the action of the statistics helper. See these constants:

`kQTSGetNameStatisticsFlag`

`kQTSDontGetDataStatisticsFlag`

`kQTSUpdateAtomsStatisticsFlag`

`kQTSGetUnitsStatisticsFlag`

outStatHelper
On entry, a pointer to a variable of type `QTSStatHelper`; on return, this variable is set to the new statistics helper.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

A statistics helper is a set of utility functions that you can use to retrieve and parse statistics from a stream component. You need to instantiate a statistics helper for every stream from which you want to gather statistics.

Special Considerations

When you are done using the statistics helper, call `QTSDisposeStatHelper` (page 1798).

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSNewStreamBuffer

Undocumented

```
OSErr QTSNewStreamBuffer (
   UInt32 inDataSize,
   SInt32 inFlags,
   QTSStreamBuffer **outStreamBuffer
);
```

Parameters

inDataSize
    Undocumented

inFlags
    Undocumented

outStreamBuffer
    Undocumented

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSPrefsAddConnectionSetting

Undocumented

```
OSErr QTSPrefsAddConnectionSetting (
   OSType protocol,
   SInt32 portID,
   UInt32 flags,
   UInt32 seed
);
```

Parameters

protocol
    A constant (see below) that identifies the connection protocol. See these constants:
        `kQTSDirectConnectHTTPProtocol`
        `kQTSDirectConnectRTSPProtocol`

portID
    Undocumented

flags
  Undocumented

seed
  Undocumented

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSPrefsAddProxySetting

Undocumented

```
OSErr QTSPrefsAddProxySetting (
   OSType proxyType,
   SInt32 portID,
   UInt32 flags,
   UInt32 seed,
   Str255 srvrURL
);
```

Parameters

proxyType
  A constant (see below) that defines the proxy type. See these constants:
    `kQTSHTTPProxyPrefsType`
    `kQTSRTSPProxyPrefsType`
    `kQTSSOCKSProxyPrefsType`
    `kQTSDontProxyDataType`

portID
  Undocumented

flags
  Undocumented

seed
  Undocumented

srvrURL
  A string containing the server's URL.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSPrefsAddProxyUserInfo

Undocumented

```
OSErr QTSPrefsAddProxyUserInfo (
   OSType proxyType,
   SInt32 flags,
   SInt32 flagsMask,
   StringPtr username,
   StringPtr password
);
```

Parameters

`proxyType`
    Undocumented

`flags`
    Undocumented

`flagsMask`
    Undocumented

`username`
    Undocumented

`password`
    Undocumented

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.1 and later.

Declared In
`QuickTimeStreaming.h`

QTSPrefsFindConnectionByType

Undocumented

```
OSErr QTSPrefsFindConnectionByType (
   OSType protocol,
   UInt32 flags,
   UInt32 flagsMask,
   QTSTransportPref **connectionHndl,
   SInt16 *count
);
```

Parameters

`protocol`
: A constant (see below) that identifies the connection protocol. See these constants:
  `kQTSDirectConnectHTTPProtocol`
  `kQTSDirectConnectRTSPProtocol`

`flags`
: *Undocumented*

`flagsMask`
: *Undocumented*

`connectionHndl`
: A handle to a `QTSTransportPref` structure.

`count`
: *Undocumented*

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSPrefsFindProxyByType

Undocumented

```
OSErr QTSPrefsFindProxyByType (
   OSType proxyType,
   UInt32 flags,
   UInt32 flagsMask,
   QTSProxyPref **proxyHndl,
   SInt16 *count
);
```

Parameters

proxyType

A constant (see below) that defines the proxy type. See these constants:

`kQTSHTTPProxyPrefsType`

`kQTSRTSPProxyPrefsType`

`kQTSSOCKSProxyPrefsType`

`kQTSDontProxyDataType`

flags

Undocumented

flagsMask

Undocumented

proxyHndl

A handle to a `QTSProxyPref` structure.

count

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.1.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeStreaming.h`

QTSPrefsFindProxyUserInfoByType

Undocumented

```
OSErr QTSPrefsFindProxyUserInfoByType (
   OSType proxyType,
   SInt32 flags,
   SInt32 flagsMask,
   StringPtr username,
   StringPtr password
);
```

Parameters

proxyType

Undocumented

flags
  Undocumented

flagsMask
  Undocumented

username
  Undocumented

password
  Undocumented

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.1 and later.

Declared In
`QuickTimeStreaming.h`

QTSPrefsGetActiveConnection

Undocumented

```
OSErr QTSPrefsGetActiveConnection (
   OSType protocol,
   QTSTransportPref *connectInfo
);
```

Parameters

protocol
  A constant (see below) that identifies the connection protocol. See these constants:
  `kQTSDirectConnectHTTPProtocol`
  `kQTSDirectConnectRTSPProtocol`

connectInfo
  A pointer to a `QTSTransportPref` structure.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSPrefsGetInstantOnSettings

Undocumented

```
OSErr QTSPrefsGetInstantOnSettings (
   QTSInstantOnPref *outPref,
   SInt32 inFlags
);
```

Parameters

outPref

A pointer to a `QTSInstantOnPref` data structure.

inFlags

Undocumented

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 6.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`QuickTimeStreaming.h`

QTSPrefsGetNoProxyURLs

Undocumented

```
OSErr QTSPrefsGetNoProxyURLs (
   QTSNoProxyPref **noProxyHndl
);
```

Parameters

noProxyHndl

A handle to a `QTSNoProxyPref` structure.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.1.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeStreaming.h`

QTSPrefsSetInstantOnSettings

Undocumented

```
OSErr QTSPrefsSetInstantOnSettings (
   QTSInstantOnPref *inPref,
   SInt32 inFlags
);
```

Parameters

inPref
: A pointer to a `QTSInstantOnPref` data structure.

inFlags
: *Undocumented*

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222), as well as in the function result. See `Error Codes`.

Version Notes
Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`QuickTimeStreaming.h`

QTSPrefsSetNoProxyURLs

Undocumented

```
OSErr QTSPrefsSetNoProxyURLs (
   char *urls,
   UInt32 flags,
   UInt32 seed
);
```

Parameters

urls
: A pointer to URL strings.

flags
: *Undocumented*

seed
: *Undocumented*

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSPresAddSourcer

Undocumented

```
OSErr QTSPresAddSourcer (
   QTSPresentation inPresentation,
   QTSStream inStream,
   ComponentInstance inSourcer,
   SInt32 inFlags
);
```

Parameters

inPresentation
Undocumented

inStream
Undocumented

inSourcer
Undocumented

inFlags
Undocumented

Return Value
See Error Codes. Returns noErr if there is no error.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
QuickTimeStreaming.h

QTSPresExport

Undocumented

```
OSErr QTSPresExport (
   QTSPresentation inPresentation,
   QTSStream inStream,
   QTSExportParams *inExportParams
);
```

Parameters

inPresentation
Undocumented

inStream
Undocumented

inExportParams
Undocumented

Return Value
You can access Movie Toolbox error returns through GetMoviesError (page 221) and GetMoviesStickyError (page 222), as well as in the function result. See Error Codes.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSPresGetActiveSegment

Undocumented

```
OSErr QTSPresGetActiveSegment (
   QTSPresentation inPresentation,
   QTSStream inStream,
   TimeValue64 *outStartTime,
   TimeValue64 *outDuration
);
```

Parameters

inPresentation
A pointer to a `QTSPresentationRecord` structure.

inStream
A pointer to a `QTSStreamRecord` structure that defines a stream.

outStartTime
Undocumented

outDuration
Undocumented

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSPresGetClip

Gets the clipping region for a streaming presentation.

```
OSErr QTSPresGetClip (
   QTSPresentation inPresentation,
   QTSStream inStream,
   RgnHandle *outClip
);
```

Parameters

inPresentation

A pointer to a `QTSPresentationRecord` structure that defines a presentation.

inStream

A pointer to a `QTSStreamRecord` structure that defines a stream.

outClip

A pointer to a handle to a `MacRegion` structure that defines a clipping region.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeStreaming.h`

QTSPresGetDimensions

Gets the dimensions of a streaming presentation.

```
OSErr QTSPresGetDimensions (
   QTSPresentation inPresentation,
   QTSStream inStream,
   Fixed *outWidth,
   Fixed *outHeight
);
```

Parameters

inPresentation

A pointer to a `QTSPresentationRecord` structure that defines a presentation.

inStream

A pointer to a `QTSStreamRecord` structure that defines a stream.

outWidth

A pointer to the width in pixels.

outHeight

A pointer to the height in pixels.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSPresGetEnable

Determines whether or not a presentation is enabled.

```
OSErr QTSPresGetEnable (
   QTSPresentation inPresentation,
   QTSStream inStream,
   Boolean *outEnableMode
);
```

Parameters

inPresentation
: A pointer to a `QTSPresentationRecord` structure.

inStream
: A pointer to a `QTSStreamRecord` structure that defines a stream.

outEnableMode
: A pointer to a Boolean that is TRUE if the presentation is enabled, FALSE otherwise.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSPresGetFlags

Gets the flags currently set for a presentation.

```
OSErr QTSPresGetFlags (
   QTSPresentation inPresentation,
   SInt32 *outFlags
);
```

Parameters

inPresentation
: A pointer to a `QTSPresentationRecord` structure that defines a presentation.

outFlags

On entry, the address of a variable of type SInt32; on return, this variable is set to the current flags (see below) for the specified presentation. See these constants:

`kQTSAutoModeFlag`

`kQTSDontShowStatusFlag`

`kQTSSendMediaFlag`

`kQTSReceiveMediaFlag`

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSPresGetGraphicsMode

Gets the graphics mode and blend color in use for video display by a stream or presentation.

```
OSErr QTSPresGetGraphicsMode (
   QTSPresentation inPresentation,
   QTSStream inStream,
   short *outMode,
   RGBColor *outOpColor
);
```

Parameters

inPresentation

A pointer to a `QTSPresentationRecord` structure that defines a presentation. If you want the graphics mode for a specific stream, pass the value `kQTSInvalidPresentation`.

inStream

A pointer to a `QTSStreamRecord` structure that defines a stream. If you want the graphics mode for the presentation as a whole, pass the value `kQTSAllStreams`.

outMode

On entry, a pointer to a short integer; on return, this variable is set to the graphics mode of the specified presentation or stream. See `Graphics Transfer Modes`.

outOpColor

On entry, the address of an `RGBColor` structure; on return, this structure is filled in with information about the color used for blending and transparent operations. The stream handler passes this color to QuickDraw as appropriate when you draw in `addPin`, `subPin`, `blend`, or `transparent` mode.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSPresGetGWorld

Gets the graphics port and graphics device in use by a stream or presentation.

```
OSErr QTSPresGetGWorld (
   QTSPresentation inPresentation,
   QTSStream inStream,
   CGrafPtr *outGWorld,
   GDHandle *outGDHandle
);
```

Parameters

inPresentation
A pointer to a `QTSPresentationRecord` structure that defines a presentation. If you want the graphics mode for a specific stream, pass the value `kQTSInvalidPresentation`.

inStream
A pointer to a `QTSStreamRecord` structure that defines a stream. If you want the graphics mode for the presentation as a whole, pass the value `kQTSAllStreams`.

outGWorld
On entry, the address of a variable of type `CGrafPtr`; on return, this variable is set to a pointer to a `CGrafPort` structure that defines the offscreen graphics world, color graphics port, or basic graphics port in use by the specified presentation or stream.

outGDHandle
On entry, the address of a variable of type `GDHandle`; on return, this variable is set to the handle of a `GDevice` structure.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSPresGetIndSourcer

Undocumented

```
OSErr QTSPresGetIndSourcer (
   QTSPresentation inPresentation,
   QTSStream inStream,
   UInt32 inIndex,
   ComponentInstance *outSourcer
);
```

Parameters

inPresentation

  Undocumented

inStream

  Undocumented

inIndex

  Undocumented

outSourcer

  Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeStreaming.h`

QTSPresGetIndStream

Get a stream associated with a presentation, based on its index number.

```
QTSStream QTSPresGetIndStream (
   QTSPresentation inPresentation,
   UInt32 inIndex
);
```

Parameters

inPresentation

  A pointer to a `QTSPresentationRecord` structure.

inIndex

  The index number of the stream.

Return Value

A pointer to a `QTSStreamRecord` structure.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSPresGetInfo

Gets information about a presentation or stream.

```
OSErr QTSPresGetInfo (
   QTSPresentation inPresentation,
   QTSStream inStream,
   OSType inSelector,
   void *ioParam
);
```

Parameters

`inPresentation`
A pointer to a `QTSPresentationRecord` structure that defines a presentation. If you want information for a specific stream, pass the value `kQTSInvalidPresentation`.

`inStream`
A pointer to a `QTSStreamRecord` structure that defines a stream. If you want information for the presentation as a whole, pass the value `kQTSAllStreams`.

inSelector

A constant (see below) that defines the information to be retrieved. See these constants:

`kQTSGetURLLink`
`kQTSTargetBufferDurationInfo`
`kQTSTargetBufferDurationInfo`
`kQTSDurationInfo`
`kQTSSourceTrackIDInfo`
`kQTSSourceLayerInfo`
`kQTSSourceLanguageInfo`
`kQTSSourceTrackFlagsInfo`
`kQTSSourceDimensionsInfo`
`kQTSSourceVolumesInfo`
`kQTSSourceMatrixInfo`
`kQTSSourceClipRectInfo`
`kQTSSourceGraphicsModeInfo`
`kQTSSourceScaleInfo`
`kQTSSourceBoundingRectInfo`
`kQTSSourceUserDataInfo`
`kQTSSourceInputMapInfo`
`kQTSStatisticsInfo`
`kQTSMinStatusDimensionsInfo`
`kQTSNormalStatusDimensionsInfo`
`kQTSTotalDataRateInfo`
`kQTSTotalDataRateInInfo`
`kQTSTotalDataRateOutInfo`
`kQTSLostPercentInfo`
`kQTSMediaTypeInfo`
`kQTSNameInfo`
`kQTSCanHandleSendDataType`
`kQTSAnnotationsInfo`

ioParam

A pointer to the retrieved information in the format shown below.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSPresGetMatrix

Gets the transformation matrix in use for the graphic display of a stream or presentation.

```
OSErr QTSPresGetMatrix (
   QTSPresentation inPresentation,
   QTSStream inStream,
   MatrixRecord *outMatrix
);
```

Parameters

inPresentation

A pointer to a `QTSPresentationRecord` structure that defines a presentation. If you want to get the matrix for a specific stream, pass the value `kQTSInvalidPresentation`.

inStream

A pointer to a `QTSStreamRecord` structure that defines a stream. If you want to get the matrix for the presentation as a whole, pass the value `kQTSAllStreams`.

outMatrix

On entry, the address of a `MatrixRecord` structure; on return, this structure is filled with the transformation matrix in use by the stream handler. Note that the matrix passed back is the one last set by `QTSPresSetMatrix` (page 1856), regardless of any additional matrixes that might have been used.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeStreaming.h`

QTSPresGetNotificationProc

Gets the notification callback of a presentation.

```
OSErr QTSPresGetNotificationProc (
   QTSPresentation inPresentation,
   QTSNotificationUPP *outNotificationProc,
   void **outRefCon
);
```

Parameters

inPresentation

A pointer to a `QTSPresentationRecord` structure.

outNotificationProc

A pointer to a Universal Procedure Pointer that accesses a `QTSNotificationProc` callback. The callback acts as a back channel from a presentation to its creator. The presentation sends notification of various events, such as a presentation, ending, or acknowledgment of a preroll request.

outRefCon

A handle to a constant to be passed to your `QTSNotificationProc`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeStreaming.h`

QTSPresGetNumSourcers

Undocumented

```
UInt32 QTSPresGetNumSourcers (
   QTSPresentation inPresentation,
   QTSStream inStream
);
```

Parameters

inPresentation

Undocumented

inStream

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeStreaming.h`

QTSPresGetNumStreams

Undocumented

```
UInt32 QTSPresGetNumStreams (
   QTSPresentation inPresentation
);
```

Parameters

inPresentation

A pointer to a `QTSPresentationRecord` structure.

Return Value
Undocumented

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSPresGetPicture

Undocumented

```
OSErr QTSPresGetPicture (
   QTSPresentation inPresentation,
   QTSStream inStream,
   PicHandle *outPicture
);
```

Parameters

`inPresentation`
A pointer to a `QTSPresentationRecord` structure.

`inStream`
A pointer to a `QTSStreamRecord` structure that defines a stream.

`outPicture`
Undocumented

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSPresGetPlayHints

Undocumented

```
OSErr QTSPresGetPlayHints (
   QTSPresentation inPresentation,
   QTSStream inStream,
   SInt32 *outFlags
);
```

Parameters

inPresentation

A pointer to a `QTSPresentationRecord` structure.

inStream

A pointer to a `QTSStreamRecord` structure that defines a stream.

outFlags

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeStreaming.h`

QTSPresGetPreferredRate

Undocumented

```
OSErr QTSPresGetPreferredRate (
   QTSPresentation inPresentation,
   Fixed *outRate
);
```

Parameters

inPresentation

A pointer to a `QTSPresentationRecord` structure.

outRate

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeStreaming.h`

QTSPresGetPresenting

Determines whether presenting is enabled or disabled for a presentation or stream.

```
OSErr QTSPresGetPresenting (
   QTSPresentation inPresentation,
   QTSStream inStream,
   Boolean *outPresentingMode
);
```

Parameters

inPresentation
: A pointer to a `QTSPresentationRecord` structure that defines a presentation. If you want to get the presenting state for a specific stream, pass the value `kQTSInvalidPresentation`.

inStream
: A pointer to a `QTSStreamRecord` structure that defines a stream. If you want to get the presenting state for the presentation as a whole, pass the value `kQTSAllStreams`.

outPresentingMode
: A pointer to a Boolean that is TRUE if presenting is enabled, FALSE if it is disabled.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSPresGetSettings

Undocumented

```
OSErr QTSPresGetSettings (
   QTSPresentation inPresentation,
   QTSStream inStream,
   QTAtomContainer *outSettings,
   SInt32 inFlags
);
```

Parameters

inPresentation
: *Undocumented*

inStream
: *Undocumented*

outSettings
: *Undocumented*

inFlags
: *Undocumented*

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSPresGetSettingsAsText

Undocumented

```
OSErr QTSPresGetSettingsAsText (
   QTSPresentation inPresentation,
   QTSStream inStream,
   SInt32 inFlags,
   OSType inSettingsType,
   Handle *outText,
   QTSPanelFilterUPP inPanelFilterProc,
   void *inPanelFilterProcRefCon
);
```

Parameters

`inPresentation`
    Undocumented

`inStream`
    Undocumented

`inFlags`
    Undocumented

`inSettingsType`
    Undocumented

`outText`
    Undocumented

`inPanelFilterProc`
    Undocumented

`inPanelFilterProcRefCon`
    Undocumented

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.1 and later.

Declared In
`QuickTimeStreaming.h`

QTSPresGetTimeBase

Undocumented

```
OSErr QTSPresGetTimeBase (
   QTSPresentation inPresentation,
   TimeBase *outTimeBase
);
```

Parameters

`inPresentation`
A pointer to a `QTSPresentationRecord` structure.

`outTimeBase`
Undocumented

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSPresGetTimeScale

Undocumented

```
OSErr QTSPresGetTimeScale (
   QTSPresentation inPresentation,
   TimeScale *outTimeScale
);
```

Parameters

`inPresentation`
A pointer to a `QTSPresentationRecord` structure.

`outTimeScale`
Undocumented

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSPresGetVolumes

Gets the sound volume levels of a stream or presentation.

```
OSErr QTSPresGetVolumes (
   QTSPresentation inPresentation,
   QTSStream inStream,
   short *outLeftVolume,
   short *outRightVolume
);
```

Parameters

`inPresentation`
A pointer to a `QTSPresentationRecord` structure that defines a presentation. If you want to get the volumes for a specific stream, pass the value `kQTSInvalidPresentation`.

`inStream`
A pointer to a `QTSStreamRecord` structure that defines a stream. If you want to get the volumes for the presentation as a whole, pass the value `kQTSAllStreams`.

`outLeftVolume`
On exit, the volume level of the left channel of the stream or presentation. The values returned may range from 0x0000 (silence) to 0x0100 (full volume).

`outRightVolume`
On exit, the volume level of the right channel of the stream or presentation. The values returned may range from 0x0000 (silence) to 0x0100 (full volume).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSPresHasCharacteristic

Undocumented

```
OSErr QTSPresHasCharacteristic (
   QTSPresentation inPresentation,
   QTSStream inStream,
   OSType inCharacteristic,
   Boolean *outHasIt
);
```

Parameters

inPresentation
  A pointer to a `QTSPresentationRecord` structure.

inStream
  A pointer to a `QTSStreamRecord` structure that defines a stream.

inCharacteristic
  Undocumented

outHasIt
  Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeStreaming.h`

QTSPresIdle

Undocumented

```
void QTSPresIdle (
   QTSPresentation inPresentation,
   QTSPresIdleParams *ioParams
);
```

Parameters

inPresentation
  A pointer to a `QTSPresentationRecord` structure.

ioParams
  Undocumented

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeStreaming.h`

QTSPresInvalidateRegion

Undocumented

```
OSErr QTSPresInvalidateRegion (
   QTSPresentation inPresentation,
   RgnHandle inRegion
);
```

Parameters

inPresentation
: A pointer to a `QTSPresentationRecord` structure.

inRegion
: *Undocumented*

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeStreaming.h`

QTSPresNewStream

Undocumented

```
OSErr QTSPresNewStream (
   QTSPresentation inPresentation,
   OSType inDataType,
   const void *inData,
   UInt32 inDataLength,
   SInt32 inFlags,
   QTSStream *outStream
);
```

Parameters

inPresentation
: A pointer to a `QTSPresentationRecord` structure.

inDataType
: *Undocumented*

inData
: *Undocumented*

inDataLength
: *Undocumented*

inFlags
: *Undocumented*

outStream
: A pointer to a `QTSStreamRecord` structure that defines a stream.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSPresPreroll

Undocumented

```
OSErr QTSPresPreroll (
   QTSPresentation inPresentation,
   QTSStream inStream,
   UInt32 inTimeValue,
   Fixed inRate,
   SInt32 inFlags
);
```

Parameters

`inPresentation`
A pointer to a `QTSPresentationRecord` structure.

`inStream`
A pointer to a `QTSStreamRecord` structure that defines a stream.

`inTimeValue`
Undocumented

`inRate`
Undocumented

`inFlags`
Undocumented

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSPresPreroll64

Undocumented

```
OSErr QTSPresPreroll64 (
   QTSPresentation inPresentation,
   QTSStream inStream,
   const TimeValue64 *inPrerollTime,
   Fixed inRate,
   SInt32 inFlags
);
```

Parameters

inPresentation
: A pointer to a `QTSPresentationRecord` structure.

inStream
: A pointer to a `QTSStreamRecord` structure that defines a stream.

inPrerollTime
: *Undocumented*

inRate
: *Undocumented*

inFlags
: *Undocumented*

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.1.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeStreaming.h`

QTSPresPreview

Undocumented

```
OSErr QTSPresPreview (
   QTSPresentation inPresentation,
   QTSStream inStream,
   const TimeValue64 *inTimeValue,
   Fixed inRate,
   SInt32 inFlags
);
```

Parameters

inPresentation
: *Undocumented*

inStream
: *Undocumented*

inTimeValue
: *Undocumented*

`inRate`
Undocumented

`inFlags`
Undocumented

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSPresRemoveSourcer

Undocumented

```
OSErr QTSPresRemoveSourcer (
   QTSPresentation inPresentation,
   QTSStream inStream,
   ComponentInstance inSourcer,
   SInt32 inFlags
);
```

Parameters

`inPresentation`
Undocumented

`inStream`
Undocumented

`inSourcer`
Undocumented

`inFlags`
Undocumented

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSPresSetActiveSegment

Undocumented

```
OSErr QTSPresSetActiveSegment (
   QTSPresentation inPresentation,
   QTSStream inStream,
   const TimeValue64 *inStartTime,
   const TimeValue64 *inDuration
);
```

Parameters

inPresentation

A pointer to a `QTSPresentationRecord` structure.

inStream

A pointer to a `QTSStreamRecord` structure that defines a stream.

inStartTime

Undocumented

inDuration

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.1.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeStreaming.h`

QTSPresSetClip

Undocumented

```
OSErr QTSPresSetClip (
   QTSPresentation inPresentation,
   QTSStream inStream,
   RgnHandle inClip
);
```

Parameters

inPresentation

A pointer to a `QTSPresentationRecord` structure.

inStream

A pointer to a `QTSStreamRecord` structure that defines a stream.

inClip

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSPresSetDimensions

Undocumented

```
OSErr QTSPresSetDimensions (
   QTSPresentation inPresentation,
   QTSStream inStream,
   Fixed inWidth,
   Fixed inHeight
);
```

Parameters

`inPresentation`
A pointer to a `QTSPresentationRecord` structure.

`inStream`
A pointer to a `QTSStreamRecord` structure that defines a stream.

`inWidth`
Undocumented

`inHeight`
Undocumented

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSPresSetEnable

Undocumented

```
OSErr QTSPresSetEnable (
   QTSPresentation inPresentation,
   QTSStream inStream,
   Boolean inEnableMode
);
```

Parameters

`inPresentation`
A pointer to a `QTSPresentationRecord` structure.

inStream
: A pointer to a `QTSStreamRecord` structure that defines a stream.

inEnableMode
: *Undocumented*

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSPresSetFlags

Undocumented

```
OSErr QTSPresSetFlags (
   QTSPresentation inPresentation,
   SInt32 inFlags,
   SInt32 inFlagsMask
);
```

Parameters

inPresentation
: A pointer to a `QTSPresentationRecord` structure.

inFlags
: *Undocumented*

inFlagsMask
: *Undocumented*

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSPresSetGraphicsMode

Sets the graphics transfer mode for a streaming presentation.

```
OSErr QTSPresSetGraphicsMode (
   QTSPresentation inPresentation,
   QTSStream inStream,
   short inMode,
   const RGBColor *inOpColor
);
```

Parameters

inPresentation
: A pointer to a `QTSPresentationRecord` structure.

inStream
: A pointer to a `QTSStreamRecord` structure that defines a stream.

inMode
: A short integer; see `Graphics Transfer Modes`.

inOpColor
: A pointer to an `RGBColor` structure. This is the blend value for blends and the transparent color for transparent operations. The toolbox supplies this value to QuickDraw when you draw in `addPin`, `subPin`, `blend`, `transparent`, or `graphicsModeStraightAlphaBlend` mode.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSPresSetGWorld

Undocumented

```
OSErr QTSPresSetGWorld (
   QTSPresentation inPresentation,
   QTSStream inStream,
   CGrafPtr inGWorld,
   GDHandle inGDHandle
);
```

Parameters

inPresentation
: A pointer to a `QTSPresentationRecord` structure.

inStream
: A pointer to a `QTSStreamRecord` structure that defines a stream.

inGWorld
: *Undocumented*

inGDHandle
: *Undocumented*

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSPresSetInfo

Sets information for a presentation or stream.

```
OSErr QTSPresSetInfo (
   QTSPresentation inPresentation,
   QTSStream inStream,
   OSType inSelector,
   void *ioParam
);
```

Parameters

inPresentation
A pointer to a `QTSPresentationRecord` structure that defines a presentation. If you want to set information for a specific stream, pass the value `kQTSInvalidPresentation`.

inStream
A pointer to a `QTSStreamRecord` structure that defines a stream. If you want to set information for the presentation as a whole, pass the value `kQTSAllStreams`.

inSelector
A constant (see below) that defines the type of information to be set. See these constants:

`kQTSGetURLLink`

`kQTSTargetBufferDurationInfo`

`kQTSDurationInfo`

`kQTSSourceTrackIDInfo`

`kQTSSourceLayerInfo`

`kQTSSourceLanguageInfo`

`kQTSSourceTrackFlagsInfo`

`kQTSSourceDimensionsInfo`

`kQTSSourceVolumesInfo`

`kQTSSourceMatrixInfo`

`kQTSSourceClipRectInfo`

`kQTSSourceGraphicsModeInfo`

`kQTSSourceScaleInfo`

`kQTSSourceBoundingRectInfo`

`kQTSSourceUserDataInfo`

`kQTSSourceInputMapInfo`

ioParam

A pointer to the information to be set in the format shown below.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeStreaming.h`

QTSPresSetMatrix

Sets the transformation matrix to be used by the graphic display of a stream or presentation.

```
OSErr QTSPresSetMatrix (
   QTSPresentation inPresentation,
   QTSStream inStream,
   const MatrixRecord *inMatrix
);
```

Parameters

inPresentation

A pointer to a `QTSPresentationRecord` structure that defines a presentation. If you want to set the matrix for a specific stream, pass the value `kQTSInvalidPresentation`.

inStream

A pointer to a `QTSStreamRecord` structure that defines a stream. If you want to set the matrix for the presentation as a whole, pass the value `kQTSAllStreams`.

inMatrix

A pointer to a `MatrixRecord` structure.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeStreaming.h`

QTSPresSetNotificationProc

Sets the notification callback for a presentation.

```
OSErr QTSPresSetNotificationProc (
   QTSPresentation inPresentation,
   QTSNotificationUPP inNotificationProc,
   void *inRefCon
);
```

Parameters

inPresentation
A pointer to a QTSPresentationRecord structure.

inNotificationProc
A Universal Procedure Pointer that accesses a QTSNotificationProc callback. The callback acts as a back channel from a presentation to its creator. The presentation sends notification of various events, such as a presentation, ending, or acknowledgment of a preroll request.

inRefCon
A pointer to data to be passed to your QTSNotificationProc.

Return Value
See Error Codes. Returns noErr if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
QuickTimeStreaming.h

QTSPresSetPlayHints

Undocumented

```
OSErr QTSPresSetPlayHints (
   QTSPresentation inPresentation,
   QTSStream inStream,
   SInt32 inFlags,
   SInt32 inFlagsMask
);
```

Parameters

inPresentation
A pointer to a QTSPresentationRecord structure that defines a presentation. If you want to set the play hints for a specific stream, pass the value kQTSInvalidPresentation.

inStream
A pointer to a QTSStreamRecord structure that defines a stream. If you want to set the play hints for the presentation as a whole, pass the value kQTSAllStreams.

inFlags
Undocumented

inFlagsMask
Undocumented

Return Value
See Error Codes. Returns noErr if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSPresSetPreferredRate

Undocumented

```
OSErr QTSPresSetPreferredRate (
   QTSPresentation inPresentation,
   Fixed inRate,
   SInt32 inFlags
);
```

Parameters

`inPresentation`
A pointer to a `QTSPresentationRecord` structure.

`inRate`
Undocumented

`inFlags`
Undocumented

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSPresSetPresenting

Enables or disables presentation of a stream to the user.

```
OSErr QTSPresSetPresenting (
   QTSPresentation inPresentation,
   QTSStream inStream,
   Boolean inPresentingMode
);
```

Parameters

`inPresentation`
A pointer to a `QTSPresentationRecord` structure that defines a presentation. If you want to enable or disable the presentation for a specific stream, pass the value `kQTSInvalidPresentation`.

inStream
A pointer to a `QTSStreamRecord` structure that defines a stream. If you want to enable or disable the presentation as a whole, pass the value `kQTSAllStreams`.

inPresentingMode
Pass TRUE to enable the presentation, FALSE to disable it.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSPresSetSettings

Undocumented

```
OSErr QTSPresSetSettings (
   QTSPresentation inPresentation,
   QTSStream inStream,
   QTAtomSpecPtr inSettings,
   SInt32 inFlags
);
```

Parameters

inPresentation
Undocumented

inStream
Undocumented

inSettings
Undocumented

inFlags
Undocumented

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSPresSettingsDialog

Undocumented

```
OSErr QTSPresSettingsDialog (
   QTSPresentation inPresentation,
   QTSStream inStream,
   SInt32 inFlags,
   QTSModalFilterUPP inFilterProc,
   void *inFilterProcRefCon
);
```

Parameters

inPresentation
: *Undocumented*

inStream
: *Undocumented*

inFlags
: *Undocumented*

inFilterProc
: *Undocumented*

inFilterProcRefCon
: *Undocumented*

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSPresSettingsDialogWithFilters

Undocumented

```
OSErr QTSPresSettingsDialogWithFilters (
   QTSPresentation inPresentation,
   QTSStream inStream,
   SInt32 inFlags,
   QTSModalFilterUPP inFilterProc,
   void *inFilterProcRefCon,
   QTSPanelFilterUPP inPanelFilterProc,
   void *inPanelFilterProcRefCon
);
```

Parameters

inPresentation
: *Undocumented*

inStream
   Undocumented

inFlags
   Undocumented

inFilterProc
   Undocumented

inFilterProcRefCon
   Undocumented

inPanelFilterProc
   Undocumented

inPanelFilterProcRefCon
   Undocumented

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.1 and later.

Declared In
`QuickTimeStreaming.h`

QTSPresSetVolumes

Sets the sound volume levels of a stream or presentation.

```
OSErr QTSPresSetVolumes (
   QTSPresentation inPresentation,
   QTSStream inStream,
   short inLeftVolume,
   short inRightVolume
);
```

Parameters

inPresentation
   A pointer to a `QTSPresentationRecord` structure that defines a presentation. If you want to set the volume of a specific stream, pass the value `kQTSInvalidPresentation`.

inStream
   A pointer to a `QTSStreamRecord` structure that defines a stream. If you want to set the volume of the presentation as a whole, pass the value `kQTSAllStreams`.

inLeftVolume
   The volume level to be set for the left channel of the stream or presentation. The values may range from 0x0000 (silence) to 0x0100 (full volume).

inRightVolume
   The volume level to be set for the right channel of the stream or presentation. The values may range from 0x0000 (silence) to 0x0100 (full volume).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSPresSkipTo

Requests that a presentation skip to a given point, specified by a time value.

```
OSErr QTSPresSkipTo (
   QTSPresentation inPresentation,
   UInt32 inTimeValue
);
```

Parameters

inPresentation

A pointer to a `QTSPresentationRecord` structure.

inTimeValue

The time value to skip to, expressed in the time scale of the presentation.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSPresSkipTo64

Requests that a streaming presentation skip to a given point, specified by a 64-bit time value.

```
OSErr QTSPresSkipTo64 (
   QTSPresentation inPresentation,
   const TimeValue64 *inTimeValue
);
```

Parameters

inPresentation

A pointer to a `QTSPresentationRecord` structure.

inTimeValue
A pointer to a signed 64-bit integer that contains the time value to skip to, expressed in the time scale of the presentation.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSPresStart

Starts a streaming presentation or a stream.

```
OSErr QTSPresStart (
   QTSPresentation inPresentation,
   QTSStream inStream,
   SInt32 inFlags
);
```

Parameters

inPresentation
A pointer to a `QTSPresentationRecord` structure that defines a presentation. If you want to start a specific stream, pass the value `kQTSInvalidPresentation`.

inStream
A pointer to a `QTSStreamRecord` structure that defines a stream. If you want to start the presentation as a whole, pass the value `kQTSAllStreams`.

inFlags
Flags (see below) that govern the starting of the presentation or stream. See these constants:

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
If `QTSPresPreroll` (page 1848) has not been called, QuickTime must set up the streams and do everything that would have been done in preroll. If the presentation has already been prerolled, it should be ready to start immediately.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSPresStop

Stops a streaming presentation or stream.

```
OSErr QTSPresStop (
   QTSPresentation inPresentation,
   QTSStream inStream,
   SInt32 inFlags
);
```

Parameters

inPresentation
A pointer to a `QTSPresentationRecord` structure that defines a presentation. If you want to stop a specific stream, pass the value `kQTSInvalidPresentation`.

inStream
A pointer to a `QTSStreamRecord` structure that defines a stream. If you want to stop the presentation as a whole, pass the value `kQTSAllStreams`. All audio and video output will cease.

inFlags
Flags that govern the stopping of the presentation or stream. No flags are currently defined.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSReleaseMemPtr

Disposes of a pointer to a streaming buffer that will be recirculated.

```
void QTSReleaseMemPtr (
   QTSMemPtr inMemPtr,
   SInt32 inFlags
);
```

Parameters

inMemPtr
A pointer to an opaque structure.

inFlags
Undocumented

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSSetNetworkAppName

Sets the name of a streaming network application.

```
OSErr QTSSetNetworkAppName (
   const char *inAppName,
   SInt32 inFlags
);
```

Parameters

inAppName
: A pointer to a string containing the application's name.

inFlags
: A flag (see below) that determines whether the name is a full pathname. See these constants:
  `kQTSNetworkAppNameIsFullNameFlag`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.1.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeStreaming.h`

QTSSourcerGetEnable

Undocumented

```
ComponentResult QTSSourcerGetEnable (
   QTSSourcer inSourcer,
   Boolean *outEnableMode,
   SInt32 inFlags
);
```

Parameters

inSourcer
: *Undocumented*

outEnableMode
: *Undocumented*

inFlags
: *Undocumented*

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

QTSSourcerGetInfo

Undocumented

```
ComponentResult QTSSourcerGetInfo (
   QTSSourcer inSourcer,
   OSType inSelector,
   void *ioParams
);
```

Parameters

inSourcer
: *Undocumented*

inSelector
: *Undocumented*

ioParams
: *Undocumented*

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

QTSSourcerGetTimeScale

Undocumented

```
ComponentResult QTSSourcerGetTimeScale (
   QTSSourcer inSourcer,
   TimeScale *outTimeScale
);
```

Parameters

inSourcer
: *Undocumented*

outTimeScale
: *Undocumented*

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

QTSSourcerIdle

Undocumented

```
ComponentResult QTSSourcerIdle (
   QTSSourcer inSourcer,
   const TimeValue64 *inTime,
   SInt32 inFlags,
   SInt32 *outFlags
);
```

Parameters

inSourcer
: *Undocumented*

inTime
: *Undocumented*

inFlags
: *Undocumented*

outFlags
: *Undocumented*

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

QTSSourcerInitialize

Undocumented

```
ComponentResult QTSSourcerInitialize (
   QTSSourcer inSourcer,
   const QTSSourcerInitParams *inInitParams
);
```

Parameters

inSourcer
: *Undocumented*

inInitParams
  Undocumented

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.1 and later.

Declared In
`QTStreamingComponents.h`

QTSSourcerSetEnable

Undocumented

```
ComponentResult QTSSourcerSetEnable (
   QTSSourcer inSourcer,
   Boolean inEnableMode,
   SInt32 inFlags
);
```

Parameters

inSourcer
  Undocumented

inEnableMode
  Undocumented

inFlags
  Undocumented

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

QTSSourcerSetInfo

Undocumented

```
ComponentResult QTSSourcerSetInfo (
   QTSSourcer inSourcer,
   OSType inSelector,
   void *ioParams
);
```

Parameters

inSourcer
  Undocumented

inSelector
  Undocumented

ioParams
  Undocumented

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

QTSSourcerSetTimeScale

Undocumented

```
ComponentResult QTSSourcerSetTimeScale (
   QTSSourcer inSourcer,
   TimeScale inTimeScale
);
```

Parameters

inSourcer
  Undocumented

inTimeScale
  Undocumented

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

QTSStatHelperGetNumStats

Gets the number of statistics that a statistic helper is reporting.

```
UInt32 QTSStatHelperGetNumStats (
   QTSStatHelper inStatHelper
);
```

Parameters

inStatHelper

A pointer to a `QTSStatHelperRecord` structure that defines the component instance of a statistics helper.

Return Value

The number of statistics.

Discussion

You can also find the number of statistics that a statistics helper is reporting by calling `QTSStatHelperResetIter` (page 1871), then calling `QTSStatHelperNext` (page 1871) iteratively until it returns FALSE and counting the iterations.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeStreaming.h`

QTSStatHelperGetStats

Tells a statistics helper to update its statistics.

```
OSErr QTSStatHelperGetStats (
   QTSStatHelper inStatHelper
);
```

Parameters

inStatHelper

A pointer to a `QTSStatHelperRecord` structure that defines the component instance of a statistics helper.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Statistics helpers update their statistics only when this function is called. You should call it at least once before calling `QTSStatHelperNext` (page 1871), to ensure that the information returned is valid and current. The normal sequence is to call this function, then call `QTSStatHelperResetIter` (page 1871), then make a series of calls to `QTSStatHelperNext` (page 1871).

Version Notes

Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSStatHelperNext

Gets the next statistic from a statistic helper.

```
Boolean QTSStatHelperNext (
   QTSStatHelper inStatHelper,
   QTSStatHelperNextParams *ioParams
);
```

Parameters

`inStatHelper`
: A pointer to a `QTSStatHelperRecord` structure that defines the component instance of a statistics helper.

`ioParams`
: On entry, a pointer to a `QTSStatHelperNextParams` structure; on return, this structure is filled in with information about the next statistic from the specified statistic helper.

Return Value
FALSE if the last statistic has been returned, TRUE otherwise.

Discussion
You need to call this function once to retrieve each statistic. The normal sequence is to call `QTSStatHelperGetStats` (page 1870), then call `QTSStatHelperResetIter` (page 1871), then make a series of calls to this function until it returns FALSE.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSStatHelperResetIter

Reset the iteration counter of a statistics helper, so the next call to QTSStatHelperNext returns the first statistic.

```
OSErr QTSStatHelperResetIter (
   QTSStatHelper inStatHelper
);
```

Parameters

`inStatHelper`
: A pointer to a `QTSStatHelperRecord` structure that defines the component instance of a statistics helper.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSStreamBufferDataInfo

Undocumented

```
void QTSStreamBufferDataInfo (
   QTSStreamBuffer *inStreamBuffer,
   unsigned char **outDataStart,
   UInt32 *outDataMaxLength
);
```

Parameters

`inStreamBuffer`
    Undocumented

`outDataStart`
    Undocumented

`outDataMaxLength`
    Undocumented

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

RTPMPDoUserDialog

Obtains media-specific settings from the user through a dialog box.

```
ComponentResult RTPMPDoUserDialog (
   RTPMediaPacketizer rtpm,
   ModalFilterUPP inFilterUPP,
   Boolean *canceled
);
```

Parameters

rtpm

The component instance of the media packetizer.

inFilterUPP

A `ModalFilterProc` callback, which may be used in a call to the Mac OS `ModalDialog` function.

canceled

On return, a Boolean which is TRUE if the user pressed the cancel button in the dialog box. If this parameter is returned TRUE, the settings prior to calling this function should be retained.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function invokes a media packetizer's modal dialog to obtain user settings. If the packetizer supports "more settings," you can put up a dialog allowing the user to enter media-specific settings. You can determine whether a packetizer has this characteristic by calling `RTPMPHasCharacteristic` (page 1880)). The settings can be obtained for storage by calling `RTPMPGetSettingsIntoAtomContainerAtAtom` (page 1878), and can be restored or set directly from an application by calling `RTPMPSetSettingsFromAtomContainerAtAtom` (page 1889).

Special Considerations

This function may be called at any time.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QTStreamingComponents.h`

RTPMPFlush

Renamed RTPMPReset.

```
ComponentResult RTPMPFlush (
   RTPMediaPacketizer rtpm,
   SInt32 inFlags,
   SInt32 *outFlags
);
```

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

QTSPketizerReassem

QTSPketizerReassem.win

qtstreaming

qtstreaming.win

Declared In

`QTStreamingComponents.h`

RTPMPGetInfo

Obtains information of various types from a media packetizer.

```
ComponentResult RTPMPGetInfo (
   RTPMediaPacketizer rtpm,
   OSType inSelector,
   void *ioParams
);
```

Parameters

`rtpm`

The component instance of the media packetizer you want information from.

`inSelector`

The selector for the type information you want (see below). See these constants:

`kRTPMPPayloadTypeInfo`

`kRTPMPRTPTimeScaleInfo`

`kRTPMPRequiredSampleDescriptionInfo`

`kRTPMPMinPayloadSize`

`kRTPMPMinPacketDuration`

`kRTPMPSuggestedRepeatPktCountInfo`

`ioParams`

A pointer to a data structure of the appropriate type to hold the information you are requesting. You need to allocate and dispose of this data structure.

Return Value

See `Error Codes`. Returns `qtsBadSelectorErr` if `inSelector` requests a selector you do not support. Returns `noErr` if there is no error.

Discussion

This function can be called at any time.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

QTSPketizerReassem

QTSPketizerReassem.win

qtstreaming

qtstreaming.win

Declared In
`QTStreamingComponents.h`

RTPMPGetMaxPacketDuration

Reads the maximum packet duration currently set for this packetizer.

```
ComponentResult RTPMPGetMaxPacketDuration (
   RTPMediaPacketizer rtpm,
   UInt32 *outMaxPacketDuration
);
```

Parameters

rtpm
: The component instance of the media packetizer.

outMaxPacketDuration
: On return, a pointer to a 32-bit integer containing the maximum packet duration, in milliseconds, that the packetizer is set to use.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
The maximum allowable packet duration can change during a presentation, so you should obtain this value immediately before using it.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

RTPMPGetMaxPacketSize

Returns the maximum packet size, in bytes, that the packetizer is set to create.

```
ComponentResult RTPMPGetMaxPacketSize (
   RTPMediaPacketizer rtpm,
   UInt32 *outMaxPacketSize
);
```

Parameters

rtpm
: The component instance of the media packetizer.

outMaxPacketSize
: On return, a pointer to a 32-bit integer containing the maximum packet size, in bytes, that the packetizer is set to create.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
The maximum allowable packet size can change during a presentation, so you should obtain this value immediately before using it.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

RTPMPGetMediaType

Obtains the data type being handled by a media packetizer.

```
ComponentResult RTPMPGetMediaType (
   RTPMediaPacketizer rtpm,
   OSType *outMediaType
);
```

Parameters

rtpm
: The component instance of the media packetizer.

outMediaType
: On return, a pointer to the media's data type, such as `VideoMediaType` or `SoundMediaType`; see `Data References`.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Special Considerations

The media's data type must be set prior to calling `RTPMPSetSampleData` (page 1887). It cannot change afterward.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

RTPMPGetPacketBuilder

Obtains the component instance of the packet builder component being used by a media packetizer.

```
ComponentResult RTPMPGetPacketBuilder (
   RTPMediaPacketizer rtpm,
   ComponentInstance *outPacketBuilder
);
```

Parameters

rtpm

The component instance of the media packetizer whose packet builder you are interested in.

outPacketBuilder

On return, a pointer to the component instance of the packet builder component in use by this media packetizer.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QTStreamingComponents.h`

RTPMPGetSettings

Undocumented

```
ComponentResult RTPMPGetSettings (
   RTPMediaPacketizer rtpm,
   QTAtomContainer *outSettings,
   SInt32 inFlags
);
```

Parameters

rtpm

Undocumented

outSettings

Undocumented

inFlags

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 6.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QTStreamingComponents.h`

RTPMPGetSettingsAsText

Return the media-specific settings of a media packetizer as text in a format presentable to the user.

```
ComponentResult RTPMPGetSettingsAsText (
   RTPMediaPacketizer rtpm,
   Handle *text
);
```

Parameters

rtpm

The component instance of a media packetizer.

text

Return a handle to a copy of your user settings in text format. The text is formatted as simple array of characters. There is no size byte or null termination. Allocate the handle to fit the text precisely.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function expects you to return your user settings as text. It should be called only if the media packetizer supports packetizer-specific settings. To determine if your media packetizer supports this function, the application may call `RTPMPHasCharacteristic` (page 1880).

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QTStreamingComponents.h`

RTPMPGetSettingsIntoAtomContainerAtAtom

Obtains the media-specific setting of a media packetizer.

```
ComponentResult RTPMPGetSettingsIntoAtomContainerAtAtom (
   RTPMediaPacketizer rtpm,
   QTAtomContainer inOutContainer,
   QTAtom inParentAtom
);
```

Parameters

rtpm

The component instance of the media packetizer.

inOutContainer

The atom container that holds the settings atom, which the caller must allocate.

inParentAtom

The atom that will hold the settings.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function should be called only if the media packetizer supports packetizer-specific settings. To determine if a media packetizer supports this function, call `RTPMPHasCharacteristic` (page 1880).

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

RTPMPGetTimeBase

Returns the time base passed to a media packetizer by RTPMPSetTimeBase.

```
ComponentResult RTPMPGetTimeBase (
   RTPMediaPacketizer rtpm,
   TimeBase *outTimeBase
);
```

Parameters

rtpm
: The component instance of your media packetizer.

outTimeBase
: A pointer to the time base passed to you by `RTPMPSetTimeBase` (page 1890).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

RTPMPGetTimeScale

Obtains the time scale in use by a media packetizer.

```
ComponentResult RTPMPGetTimeScale (
   RTPMediaPacketizer rtpm,
   TimeScale *outTimeScale
);
```

Parameters

rtpm
: The component instance of media packetizer component.

outTimeScale

On return, contains a pointer to the time scale in use by the packetizer. The time scale indicates the number of time units that pass in one second when the media is playing at a rate of 1.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QTStreamingComponents.h`

RTPMPHasCharacteristic

Determines whether a media packetizer has a particular characteristic, such as whether it supports a user settings dialog.

```
ComponentResult RTPMPHasCharacteristic (
   RTPMediaPacketizer rtpm,
   OSType inSelector,
   Boolean *outHasIt
);
```

Parameters

rtpm

The component instance of the media packetizer.

inSelector

A selector for the characteristic you want to know about. See these constants:

`kRTPMPNoSampleDataRequiredCharacteristic`

`kRTPMPHasUserSettingsDialogCharacteristic`

`kRTPMPPrefersReliableTransportCharacteristic`

`kRTPMPRequiresOutOfBandDimensionsCharacteristic`

outHasIt

On return, contains a Boolean value that is TRUE if the media packetizer has this characteristic, FALSE otherwise.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

QTSPketizerReassem

QTSPketizerReassem.win

qtstreaming

qtstreaming.win

Declared In
`QTStreamingComponents.h`

RTPMPIdle

Called periodically in your event loop to allocate time to each media packetizer.

```
ComponentResult RTPMPIdle (
   RTPMediaPacketizer rtpm,
   SInt32 inFlags,
   SInt32 *outFlags
);
```

Parameters

`rtpm`
The component instance of the media packetizer.

`inFlags`
There are currently no defined flags.

`outFlags`
On return, contains a pointer to a signed 32-bit integer that holds a flag (see below) from the packetizer. See these constants:
`kRTPMPStillProcessingData`

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
The packetizer will use this time to process the data in its buffer. If the data has not all been processed, this function returns the `kRTPMPStillProcessingData` flag. Data is placed in the buffer by `RTPMPSetSampleData` (page 1887).

Special Considerations

The packetizer may make calls to the packet builder in response to this call.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

RTPMPInitialize

Initializes a media packetizer component.

```
ComponentResult RTPMPInitialize (
   RTPMediaPacketizer rtpm,
   SInt32 inFlags
);
```

Parameters

rtpm

The component instance of the media packetizer.

inFlags

A signed 32-bit integer containing the flags (see below) you wish to pass to the packetizer at start-up. See these constants:

`kRTPMPRealtimeModeFlag`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The calling component must call this function before sending any data to a media packetizer or making any `RTPMPSet` calls. The calling component then calls `RTPMPSetSampleData` (page 1887) and `RTPMPIdle` (page 1881) repeatedly. The calling component passes sample data (obtained, for example, from `GetMediaSample` (page 1583)), to the media packetizer by calling `RTPMPSetSampleData`. If `RTPMPSetSampleData` or `RTPMPIdle` return the flag `kRTPMPStillProcessingData`, then the calling component should call `RTPMPIdle`; if not, it is free to call `RTPMPSetSampleData` again.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

QTSPketizerReassem

QTSPketizerReassem.win

qtstreaming

qtstreaming.win

Declared In

`QTStreamingComponents.h`

RTPMPPreflightMedia

Determines whether your packetizer can work with a given media type and sample description.

```
ComponentResult RTPMPPreflightMedia (
   RTPMediaPacketizer rtpm,
   OSType inMediaType,
   SampleDescriptionHandle inSampleDescription
);
```

Parameters

rtpm

The component instance of your media packetizer.

inMediaType

The media type, such as `'vide'`; see `Data References`.

inSampleDescription

A handle to the `SampleDescription` structure.

Return Value

Return `noErr` if you can packetize this type of data; return `qtsUnsupportedFeatureErr` if you cannot. See `Error Codes`.

Discussion

This function must be implemented by your packetizer. It will be called before you are asked to packetize any data.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QTStreamingComponents.h`

RTPMPReset

Allows a media packetizer to stop packetizing its current input, set its state to idle, and flush its input buffer.

```
ComponentResult RTPMPReset (
   RTPMediaPacketizer rtpm,
   SInt32 inFlags
);
```

Parameters

rtpm

The component instance of the media packetizer.

inFlags

A signed 32-bit integer containing any flags you are passing to the media packetizer. There are currently no defined flags.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

You can use this function to stop the media packetizer and flush its input buffer when you wish to stop transmitting immediately, when you are skipping forward or backward in the stream, or if the network data connection is interrupted.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

QTSPketizerReassem

QTSPketizerReassem.win

qtstreaming

qtstreaming.win

Declared In
`QTStreamingComponents.h`

RTPMPSetInfo

Sets any one of several parameters for a media packetizer.

```
ComponentResult RTPMPSetInfo (
   RTPMediaPacketizer rtpm,
   OSType inSelector,
   const void *ioParams
);
```

Parameters

`rtpm`

The component instance of the media packetizer.

`inSelector`

A selector (see below) for the type of information you wish to set. See these constants:

`kQTSSourceTrackIDInfo`

`kQTSSourceLayerInfo`

`kQTSSourceLanguageInfo`

`kQTSSourceTrackFlagsInfo`

`kQTSSourceDimensionsInfo`

`kQTSSourceVolumesInfo`

`kQTSSourceMatrixInfo`

`kQTSSourceClipRectInfo`

`kQTSSourceGraphicsModeInfo`

`kQTSSourceBoundingRectInfo`

`kQTSSourceScaleInfo`

`kQTSSourceUserDataInfo`

`kQTSSourceInputMapInfo`

`ioParams`

A pointer to a data structure of the appropriate type for the information you are passing.

Return Value
Return `qtsBadSelectorErr` if you do not support the selector. Return `noErr` if there is no error. See `Error Codes`.

Discussion
This function is used to pass track-level information about the media track to be packetized, such as its track ID, layer, and transformation matrix. Return `qtsBadSelectorErr` unless your packetizer is able to transmit this kind of data to your reassembler for use in the client movie.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

RTPMPSetMaxPacketDuration

Sets the maximum packet duration that the media packetizer is to use.

```
ComponentResult RTPMPSetMaxPacketDuration (
   RTPMediaPacketizer rtpm,
   UInt32 inMaxPacketDuration
);
```

Parameters

rtpm
The component instance of the media packetizer.

inMaxPacketDuration
An unsigned 32-bit integer containing the maximum packet duration in milliseconds. This value should not be smaller than the value returned from `RTPMPGetInfo` (page 1874) with the `kRTPMPMinPacketDuration` selector.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
The maximum packet duration cannot be changed during a presentation, and this function cannot be called after calling `RTPMPSetSampleData` (page 1887).

Special Considerations
If `RTPMPSetMaxPacketDuration` is not called, a default value will be used.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

RTPMPSetMaxPacketSize

Sets the maximum packet size for packets created by a media packetizer.

```
ComponentResult RTPMPSetMaxPacketSize (
   RTPMediaPacketizer rtpm,
   UInt32 inMaxPacketSize
);
```

Parameters

rtpm

The component instance of the media packetizer.

inMaxPacketSize

An unsigned 32-bit integer specifying the maximum size, in bytes, of packets to be created. This value must not be smaller than the value returned from `RTPMPGetInfo` (page 1874) with the `kRTPMPMinPayloadSize` selector. The media packetizer will not create packets larger than this value. The limit applies only to the payload data.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The maximum packet size cannot change during a presentation. Streaming will be most efficient if this value is set to the largest packet size that can traverse the network without being split. `RTPMPSetMaxPacketSize` may not be called after calling `RTPMPSetSampleData` (page 1887).

Special Considerations

If `RTPMPSetMaxPacketSize` is not called, a default value will be used.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QTStreamingComponents.h`

RTPMPSetMediaType

Sets the type of media that a media packetizer will process.

```
ComponentResult RTPMPSetMediaType (
   RTPMediaPacketizer rtpm,
   OSType inMediaType
);
```

Parameters

rtpm

The component instance of the media packetizer.

inMediaType

The media type; see `Data References`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The media type must be set prior to calling `RTPMPSetSampleData` (page 1887) and cannot change after such calls.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QTStreamingComponents.h`

RTPMPSetPacketBuilder

Selects which packet builder a media packetizer will use.

```
ComponentResult RTPMPSetPacketBuilder (
   RTPMediaPacketizer rtpm,
   ComponentInstance inPacketBuilder
);
```

Parameters

`rtpm`
: The component instance of the media packetizer.

`inPacketBuilder`
: The component instance of the packet builder component to use.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

A media packetizer always sends its output to a packet builder. The specified packet builder may assemble actual RTP packets, or it may use information about the packet to build a hint track. You must set the packet builder using this call prior to any calls to `RTPMPSetSampleData` (page 1887). You can also use this function to dynamically change the packet builder a media packetizer uses.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QTStreamingComponents.h`

RTPMPSetSampleData

Provides sample data directly to a media packetizer component.

```
ComponentResult RTPMPSetSampleData (
   RTPMediaPacketizer rtpm,
   const RTPMPSampleDataParams *inSampleData,
   SInt32 *outFlags
);
```

Parameters

rtpm

The component instance of the media packetizer.

inSampleData

A pointer to a `RTPMPSampleDataParams` structure containing the sample data you are passing. Calling this routine adds data cumulatively to any previous calls to this function. The data can contain any number of samples (1 or more), or a partial sample.

outFlags

Flags (see below) that indicate processing status. This function will return `kRTPMPWantsMoreDataFlag` if it has completed processing of all pending data. Otherwise, you must make calls to `RTPMPIdle` (page 1881) until this function no longer returns `kRTPMPStillProcessingData`. See these constants:

`kRTPMPStillProcessingData`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This routine is called to pass media data directly to a media packetizer. The packetizer will not copy this data; it will call the release callback when it is finished with it. The media packetizer may or may not make calls to the packet builder in response to this call.

Special Considerations

This call is normally followed by a series of calls to `RTPMPIdle` (page 1881), which grants time to the media packetizer in order to process the data passed by this function.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QTStreamingComponents.h`

RTPMPSetSettings

Undocumented

```
ComponentResult RTPMPSetSettings (
   RTPMediaPacketizer rtpm,
   QTAtomSpecPtr inSettings,
   SInt32 inFlags
);
```

Parameters

rtpm

Undocumented

inSettings
    Undocumented

inFlags
    Undocumented

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

RTPMPSetSettingsFromAtomContainerAtAtom

Sets the media-specific settings of a media packetizer, using an atom inside an atom container.

```
ComponentResult RTPMPSetSettingsFromAtomContainerAtAtom (
   RTPMediaPacketizer rtpm,
   QTAtomContainer inContainer,
   QTAtom inParentAtom
);
```

Parameters

rtpm
    The component instance of the media packetizer.

inContainer
    The atom container that holds the settings atom.

inParentAtom
    The atom that holds the settings.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function should be called only if the media packetizer supports packetizer-specific settings. To determine if a media packetizer supports this function, call `RTPMPHasCharacteristic` (page 1880).

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

RTPMPSetTimeBase

Tells your packetizer what time base is in use by the calling application.

```
ComponentResult RTPMPSetTimeBase (
   RTPMediaPacketizer rtpm,
   TimeBase inTimeBase
);
```

Parameters

rtpm

Component instance of your packetizer.

inTimeBase

The time base in use for this stream. You can query this time base to find out the current time in the stream.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function may be called during setup for a live transmission.

Special Considerations

Your packetizer should not rely on receiving this call.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QTStreamingComponents.h`

RTPMPSetTimeScale

Sets the time scale the media packetizer will use.

```
ComponentResult RTPMPSetTimeScale (
   RTPMediaPacketizer rtpm,
   TimeScale inTimeScale
);
```

Parameters

rtpm

The component instance of the media packetizer.

inTimeScale

The time scale to use.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The time scale is the number of time units that pass in one second when the media is playing at a rate of 1. This time scale gives meaning to the times used when calling `RTPMPSetSampleData` (page 1887).

Special Considerations

The time scale must be set before calling `RTPMPSetSampleData` (page 1887).

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QTStreamingComponents.h`

RTPPBAddPacketLiteralData

Passes literal data directly to a packet builder component.

```
ComponentResult RTPPBAddPacketLiteralData (
   RTPPacketBuilder rtpb,
   SInt32 inFlags,
   RTPPacketGroupRef inPacketGroup,
   RTPPacketRef inPacket,
   UInt8 *inData,
   UInt32 inDataLength,
   RTPPacketRepeatedDataRef *outDataRef
);
```

Parameters

rtpb

The component instance of the packet builder component.

inFlags

A signed 32-bit integer containing any flags you are passing. There are currently no defined flags.

inPacketGroup

The packet group containing the packet into which the data will be placed. This is normally a reference returned by `RTPPBBeginPacketGroup` (page 1897).

inPacket

The RTP packet into which the data will be placed. This is normally a reference returned by `RTPPBBeginPacket` (page 1896).

inData

A pointer to the data you are passing.

inDataLength

An unsigned 32-bit integer containing the length, in bytes, of the data you are passing.

outDataRef

On return, contains a pointer to a data reference. Use this reference if you wish to later tell the packet builder to use this same data again, without having to literally pass the data again. Pass in `NIL` if you do not need the packet builder to repeat the data. If you do not pass in `NIL`, you must dispose of the data explicitly by calling `RTPPBReleaseRepeatedData` (page 1904).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function will return a reference which can be used to specify the same data repeatedly without having to pass in the data again. This is done by calling `RTPPBAddPacketRepeatedData` (page 1892) with the reference which was returned by this function. For example, you can use this function to insert static header information into a packet prior to inserting media sample data. It will return a data reference you can use to insert the same static information into later packets.

Special Considerations

To specify media data to be placed in a packet, a media packetizer should call `RTPPBAddPacketSampleData` (page 1893).

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

QTSPketizerReassem

QTSPketizerReassem.win

qtstreaming

qtstreaming.win

Declared In

`QTStreamingComponents.h`

RTPPBAddPacketRepeatedData

Tells a packet builder component to insert previously-specified data into a packet.

```
ComponentResult RTPPBAddPacketRepeatedData (
   RTPPacketBuilder rtpb,
   SInt32 inFlags,
   RTPPacketGroupRef inPacketGroup,
   RTPPacketRef inPacket,
   RTPPacketRepeatedDataRef inDataRef
);
```

Parameters

rtpb

The component instance of the packet builder component.

inFlags

A signed 32-bit integer containing any flags you are passing. There are currently no defined flags.

inPacketGroup

The packet group containing the packet into which the data will be placed. This is normally a reference returned by `RTPPBBeginPacketGroup` (page 1897).

inPacket

The RTP packet into which the data will be placed. This is normally a reference returned by `RTPPBBeginPacket` (page 1896).

inDataRef

A reference to the data to repeat. This is normally a data reference returned by `RTPPBAddPacketLiteralData` (page 1891) or `RTPPBAddPacketSampleData` (page 1893).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Use this function to cause a packet builder component to repeatedly insert the same data into packets without having to pass the data each time. This is typically done to repeat static header information into a series of packets, or to insert previously-sent sample data into a redundant packet. The data is first specified by a call to `RTPPBAddPacketLiteralData` (page 1891) or `RTPPBAddPacketSampleData` (page 1893), which inserts the data the first time and returns a data reference. The data reference is then used with this function to send the data again.

Special Considerations

When you are done sending the repeated data, release the data structure by calling `RTPPBReleaseRepeatedData` (page 1904).

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QTStreamingComponents.h`

RTPPBAddPacketSampleData

Commands a packet builder component to insert media sample data into a packet.

```
ComponentResult RTPPBAddPacketSampleData (
   RTPPacketBuilder rtpb,
   SInt32 inFlags,
   RTPPacketGroupRef inPacketGroup,
   RTPPacketRef inPacket,
   RTPMPSampleDataParams *inSampleDataParams,
   UInt32 inSampleOffset,
   UInt32 inSampleDataLength,
   RTPPacketRepeatedDataRef *outDataRef
);
```

Parameters

rtpb

The component instance of the packet builder component.

inFlags

A signed 32-bit integer containing any flags you are passing. There are currently no defined flags.

inPacketGroup

The packet group containing the packet into which the data will be placed. This is normally a reference returned by `RTPPBBeginPacketGroup` (page 1897).

inPacket
: The RTP packet into which the data will be placed. This is normally a reference returned by `RTPPBBeginPacket` (page 1896).

inSampleDataParams
: A pointer to a `RTPMPSampleDataParams` structure for the sample data you are inserting.

inSampleOffset
: A 32-bit unsigned integer containing the offset into the sample media, in bytes.

inSampleDataLength
: A 32-bit unsigned integer specifying the number of bytes of media sample data to insert into the packet.

outDataRef
: On return, contains a pointer to a data reference. Use this reference if you wish to later tell the packet builder to use this same sample data again, without having to literally pass the data again. Pass in `NIL` if you do not need the packet builder to repeat the data. If you do not pass in `NIL`, you must dispose of the data explicitly by calling `RTPPBReleaseRepeatedData` (page 1904).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function will return a reference which can be used to specify the same data repeatedly without having to pass in the data again. The media packetizer specifies the offset into the media and the length of the sample to insert. You can insert data repeatedly by calling `RTPPBAddPacketRepeatedData` (page 1892) with the reference which was returned by `RTPPBAddPacketLiteralData` (page 1891).

Special Considerations

When a reference is no longer needed, it should be disposed of by using the call `RTPPBReleaseRepeatedData` (page 1904).

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTSPketizerReassem

QTSPketizerReassem.win

qtstreaming

qtstreaming.win

Declared In
`QTStreamingComponents.h`

RTPPBAddPacketSampleData64

Provides a 64-bit version of RTPPBAddPacketSampleData for large sample media.

```
ComponentResult RTPPBAddPacketSampleData64 (
   RTPPacketBuilder rtpb,
   SInt32 inFlags,
   RTPPacketGroupRef inPacketGroup,
   RTPPacketRef inPacket,
   RTPMPSampleDataParams *inSampleDataParams,
   const UInt64 *inSampleOffset,
   UInt32 inSampleDataLength,
   RTPPacketRepeatedDataRef *outDataRef
);
```

Parameters

rtpb
: The component instance of the packet builder component.

inFlags
: A signed 32-bit integer containing any flags you are passing. There are currently no defined flags.

inPacketGroup
: The packet group containing the packet into which the data will be placed. This is normally a reference returned by `RTPPBBeginPacketGroup` (page 1897).

inPacket
: The RTP packet into which the data will be placed. This is normally a reference returned by `RTPPBBeginPacket` (page 1896).

inSampleDataParams
: A pointer to a `RTPMPSampleDataParams` structure for the sample data you are inserting.

inSampleOffset
: A 64-bit unsigned integer containing the offset into the sample media, in bytes.

inSampleDataLength
: A 32-bit unsigned integer specifying the number of bytes of media sample data to insert into the packet.

outDataRef
: On return, contains a pointer to a data reference. Use this reference if you wish to later tell the packet builder to use this same sample data again, without having to literally pass the data again. Pass in `NIL` if you do not need the packet builder to repeat the data. If you do not pass in `NIL`, you must dispose of the data explicitly by calling `RTPPBReleaseRepeatedData` (page 1904).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

RTPPBAddRepeatPacket

Undocumented

```
ComponentResult RTPPBAddRepeatPacket (
   RTPPacketBuilder rtpb,
   SInt32 inFlags,
   RTPPacketGroupRef inPacketGroup,
   RTPPacketRef inPacket,
   TimeValue inTransmissionOffset,
   UInt32 inSequenceNumber
);
```

Parameters

rtpb
: The component instance of the packet builder component.

inFlags
: A signed 32-bit integer containing any flags you are passing. There are currently no defined flags.

inPacketGroup
: The packet group containing the packet into which the data will be placed. This is normally a reference returned by RTPPBBeginPacketGroup (page 1897).

inPacket
: The RTP packet into which the data will be placed. This is normally a reference returned by RTPPBBeginPacket (page 1896).

inTransmissionOffset
: *Undocumented*

inSequenceNumber
: *Undocumented*

Return Value
See Error Codes. Returns noErr if there is no error.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
QTStreamingComponents.h

RTPPBBeginPacket

Tells a packet builder to create a new packet.

```
ComponentResult RTPPBBeginPacket (
   RTPPacketBuilder rtpb,
   SInt32 inFlags,
   RTPPacketGroupRef inPacketGroup,
   UInt32 inPacketMediaDataLength,
   RTPPacketRef *outPacket
);
```

Parameters

rtpb
: The component instance of the packet builder component.

inFlags
A signed 32-bit integer containing any flags you are passing. There are currently no defined flags.

inPacketGroup
The packet group containing the new packet. This is normally a reference returned by `RTPPBBeginPacketGroup` (page 1897).

inPacketMediaDataLength
An unsigned 32-bit integer specifying the maximum length of data that will be inserted into this packet. This includes the data for all subsequent `RTPPBAddPacketLiteralData` (page 1891), `RTPPBAddPacketSampleData` (page 1893), and `RTPPBAddPacketRepeatedData` (page 1892) calls until the packet is closed. The value of this parameter may be larger, but must not be smaller, than the amount of data inserted in the packet.

outPacket
On return, contains a pointer to the packet. Use this reference to insert data into the packet.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
The media packetizer uses this function to create each new packet, before inserting any literal, repeated, or sample data. A call to `RTPPBBeginPacketGroup` (page 1897) must be made before creating the first packet in a group. Data can be inserted into the packet using `RTPPBAddPacketLiteralData` (page 1891), `RTPPBAddPacketRepeatedData` (page 1892), or `RTPPBAddPacketSampleData` (page 1893). When the packet is complete, call `RTPPBEndPacket` (page 1898).

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTSPketizerReassem

QTSPketizerReassem.win

qtstreaming

qtstreaming.win

Declared In
`QTStreamingComponents.h`

RTPPBBeginPacketGroup

Tells a packet builder to create a new packet group.

```
ComponentResult RTPPBBeginPacketGroup (
   RTPPacketBuilder rtpb,
   SInt32 inFlags,
   UInt32 inTimeStamp,
   RTPPacketGroupRef *outPacketGroup
);
```

Parameters

rtpb
: The component instance of the packet builder component.

inFlags
: A signed 32-bit integer containing any flags you are passing. There are currently no defined flags.

inTimeStamp
: A unsigned 32-bit integer containing the time stamp for this packet group.

outPacketGroup
: On return, contains a pointer to a reference to the packet group. Use this data reference when creating a new packet or inserting data into a packet that belongs to this group.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
A media packetizer creates a packet group using this function. The data reference returned by this function is then used to create a series of packets that belong to this group. The data reference is also required when inserting data into packets.

Special Considerations
When the packet group is complete, call `RTPPBEndPacketGroup` (page 1899).

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTSPketizerReassem
QTSPketizerReassem.win
qtstreaming
qtstreaming.win

Declared In
`QTStreamingComponents.h`

RTPPBEndPacket

Tells a packet builder that a packet is complete.

```
ComponentResult RTPPBEndPacket (
   RTPPacketBuilder rtpb,
   SInt32 inFlags,
   RTPPacketGroupRef inPacketGroup,
   RTPPacketRef inPacket,
   UInt32 inTransmissionTimeOffset,
   UInt32 inDuration
);
```

Parameters

rtpb
: The component instance of the packet builder component.

inFlags
: A signed 32-bit integer containing any flags you are passing. There are currently no defined flags.

inPacketGroup
: The packet group containing the new packet. This is normally a reference returned by `RTPPBBeginPacketGroup` (page 1897).

inPacket
: The RTP packet containing the data. This is normally a reference returned by `RTPPBBeginPacket` (page 1896).

inTransmissionTimeOffset
: The time offset at which the media sample data contained in this packet begins, in milliseconds. This offset is added to the RTP transmission time to determine when to send the packet.

inDuration
: The duration of this packet, specified in milliseconds.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Call this function once when each packet is complete.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTSPketizerReassem
QTSPketizerReassem.win
qtstreaming
qtstreaming.win

Declared In
`QTStreamingComponents.h`

RTPPBEndPacketGroup

Tells a packet builder component that a packet group is complete.

```
ComponentResult RTPPBEndPacketGroup (
   RTPPacketBuilder rtpb,
   SInt32 inFlags,
   RTPPacketGroupRef inPacketGroup
);
```

Parameters

rtpb
: The component instance of the packet builder component.

inFlags
: A signed 32-bit integer containing any flags you are passing. There are currently no defined flags.

inPacketGroup
: A data reference to the packet group being ended. This is normally a data reference returned by `RTPPBBeginPacketGroup` (page 1897).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function should be called when all the packets in a group are complete and the media packetizer is ready either to create a new packet group or to terminate the stream.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

QTSPketizerReassem

QTSPketizerReassem.win

qtstreaming

qtstreaming.win

Declared In

`QTStreamingComponents.h`

RTPPBGetCallback

Gets the callback used to communicate with the caller of a media packetizer.

```
ComponentResult RTPPBGetCallback (
   RTPPacketBuilder rtpb,
   RTPPBCallbackUPP *outCallback,
   void **outRefCon
);
```

Parameters

rtpb
: The component instance of the packet builder component.

outCallback
: A pointer to an `RTPPBCallbackProc` callback.

outRefCon
A handle to any data your callback needs.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

RTPPBGetInfo

Gets information about a streaming packet builder.

```
ComponentResult RTPPBGetInfo (
   RTPPacketBuilder rtpb,
   OSType inSelector,
   void *ioParams
);
```

Parameters

rtpb
The component instance of the packet builder component.

inSelector
A constant (see below) that defines the type of information to retrieve. See these constants:

ioParams
A pointer to the retrieved information.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

RTPPBGetPacketSequenceNumber

Gets the relative sequence number for a streaming packet.

```
ComponentResult RTPPBGetPacketSequenceNumber (
   RTPPacketBuilder rtpb,
   SInt32 inFlags,
   RTPPacketGroupRef inPacketGroup,
   RTPPacketRef inPacket,
   UInt32 *outSequenceNumber
);
```

Parameters

rtpb
The component instance of the packet builder component.

inFlags
Undocumented

inPacketGroup
A data reference to a packet group. This is normally a data reference returned by `RTPPBBeginPacketGroup` (page 1897).

inPacket
The RTP packet. This is normally a reference returned by `RTPPBBeginPacket` (page 1896).

outSequenceNumber
A pointer to the sequence number.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

RTPPBGetPacketTimeStampOffset

Undocumented

```
ComponentResult RTPPBGetPacketTimeStampOffset (
   RTPPacketBuilder rtpb,
   SInt32 inFlags,
   RTPPacketGroupRef inPacketGroup,
   RTPPacketRef inPacket,
   SInt32 *outTimeStampOffset
);
```

Parameters

rtpb
The component instance of the packet builder component.

inFlags
A signed 32-bit integer containing any flags you are passing. There are currently no defined flags.

inPacketGroup
The packet group containing the packet of interest. This is normally a reference returned by `RTPPBBeginPacketGroup` (page 1897).

inPacket
The RTP packet of interest. This is normally a reference returned by `RTPPBBeginPacket` (page 1896).

outTimeStampOffset
Undocumented

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

RTPPBGetSampleData

Undocumented

```
ComponentResult RTPPBGetSampleData (
   RTPPacketBuilder rtpb,
   RTPMPSampleDataParams *inParams,
   const UInt64 *inStartOffset,
   UInt8 *outDataBuffer,
   UInt32 inBytesToRead,
   UInt32 *outBytesRead,
   SInt32 *outFlags
);
```

Parameters

rtpb
The component instance of the packet builder component.

inParams
A pointer to a `RTPMPSampleDataParams` structure.

inStartOffset
Undocumented

outDataBuffer
Undocumented

inBytesToRead
Undocumented

outBytesRead
Undocumented

outFlags
Undocumented

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

RTPPBReleaseRepeatedData

Lets a packet builder deallocate data that will no longer be used.

```
ComponentResult RTPPBReleaseRepeatedData (
   RTPPacketBuilder rtpb,
   RTPPacketRepeatedDataRef inDataRef
);
```

Parameters

rtpb
The component instance of the packet builder component.

inDataRef
The data reference to the repeated data. This is normally a data reference returned by `RTPPBAddPacketLiteralData` (page 1891) or `RTPPBAddPacketSampleData` (page 1893).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
You must release the data if you have allowed `RTPPBAddPacketLiteralData` (page 1891) or `RTPPBAddPacketSampleData` (page 1893) to return a data reference, even if you have not called `RTPPBAddPacketRepeatedData` (page 1892).You must either pass `NIL` to the data reference when adding literal or sample data, or you must release the data by calling this function.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

RTPPBSetCallback

Sets the callback used to communicate with the caller of a media packetizer.

```
ComponentResult RTPPBSetCallback (
   RTPPacketBuilder rtpb,
   RTPPBCallbackUPP inCallback,
   void *inRefCon
);
```

Parameters

rtpb

The component instance of the packet builder component.

inCallback

A Universal Procedure Pointer that references an `RTPPBCallbackProc` callback.

inRefCon

A pointer to any data your callback needs.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QTStreamingComponents.h`

RTPPBSetInfo

Sets information for a streaming packet builder.

```
ComponentResult RTPPBSetInfo (
   RTPPacketBuilder rtpb,
   OSType inSelector,
   void *ioParams
);
```

Parameters

rtpb

The component instance of the packet builder component.

inSelector

A constant (see below) that defines the type of information to set. See these constants:

ioParams

A pointer to the information.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QTStreamingComponents.h`

RTPPBSetPacketSequenceNumber

Sets the relative sequence number for a streaming packet.

```
ComponentResult RTPPBSetPacketSequenceNumber (
   RTPPacketBuilder rtpb,
   SInt32 inFlags,
   RTPPacketGroupRef inPacketGroup,
   RTPPacketRef inPacket,
   UInt32 inSequenceNumber
);
```

Parameters

rtpb

The component instance of the packet builder component.

inFlags

Undocumented

inPacketGroup

A data reference to a packet group. This is normally a data reference returned by `RTPPBBeginPacketGroup` (page 1897).

inPacket

The RTP packet. This is normally a reference returned by `RTPPBBeginPacket` (page 1896).

inSequenceNumber

The sequence number to be set.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QTStreamingComponents.h`

RTPPBSetPacketTimeStampOffset

Undocumented

```
ComponentResult RTPPBSetPacketTimeStampOffset (
   RTPPacketBuilder rtpb,
   SInt32 inFlags,
   RTPPacketGroupRef inPacketGroup,
   RTPPacketRef inPacket,
   SInt32 inTimeStampOffset
);
```

Parameters

rtpb

The component instance of the packet builder component.

inFlags

A signed 32-bit integer containing any flags you are passing. There are currently no defined flags.

inPacketGroup

The packet group containing the packet of interest. This is normally a reference returned by `RTPPBBeginPacketGroup` (page 1897).

inPacket

The RTP packet of interest. This is normally a reference returned by `RTPPBBeginPacket` (page 1896).

inTimeStampOffset

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QTStreamingComponents.h`

RTPRssmAdjustPacketParams

Called by the base reassembler when it is processing a packet, allowing your packet reassembler to adjust the packet parameters before the packet is processed.

```
ComponentResult RTPRssmAdjustPacketParams (
   RTPReassembler rtpr,
   RTPRssmPacket *inPacket,
   SInt32 inFlags
);
```

Parameters

rtpr

The component instance of your packet reassembler

inPacket

A pointer to the `packet` whose parameters can be adjusted.

inFlags

A signed 32-bit integer containing any flags (see below) being passed to your packet reassembler.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Your packet reassembler can adjust the following parameters in each packet: `payloadHeaderLength`, `dataLength`, `serverEditParams`, and `chunkFlags`. If your packet reassembler does not implement this function, or takes no action, the default for these parameters will be: `payloadHeaderLength` =fixed header length that is set (default is 0); `dataLength` =`packetData` - `transportHeaderLength` - `payloadHeaderLength`; no `serverEditParams`; `chunkFlags` =0.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QTStreamingComponents.h`

RTPRssmClearCachedPackets

Forces the base reassembler to flush all packets currently queued in its lists.

```
ComponentResult RTPRssmClearCachedPackets (
   RTPReassembler rtpr,
   SInt32 inFlags
);
```

Parameters

rtpr

The component instance of the base reassembler.

inFlags

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function retains the last sequence number and related information. It is useful only when the base reassembler is operating with the `kRTPRssmQueueAndUseMarkerBitFlag` flag set; see `RTPRssmSetCapabilities` (page 1926).

Version Notes

Introduced in QuickTime 4.1. Replaces `RTPRssmFlushPackets`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QTStreamingComponents.h`

RTPRssmComputeChunkSize

Lets your packet reassembler compute the size of a chunk, based on the packet list for the chunk, using your own algorithm.

```
ComponentResult RTPRssmComputeChunkSize (
   RTPReassembler rtpr,
   RTPRssmPacket *inPacketListHead,
   SInt32 inFlags,
   UInt32 *outChunkDataSize
);
```

Parameters

rtpr

The component instance of your packet reassembler.

inPacketListHead

A pointer to the list of packets that make up this chunk.

inFlags

A signed 32-bit integer containing any flags being passed to your packet reassembler.

outChunkDataSize

You should return a pointer to an unsigned 32-bit variable containing the calculated size for this chunk.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function is called once for each packet list. Implement this function only if you need to override the base reassembler's default computation. If you do not implement this call, the base reassembler will compute the chunk size by summing the data lengths for all packets in the list.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QTStreamingComponents.h`

RTPRssmCopyDataToChunk

Lets your packet reassembler write the chunk data, based on the list of packets for the chunk, using your own algorithm.

```
ComponentResult RTPRssmCopyDataToChunk (
   RTPReassembler rtpr,
   RTPRssmPacket *inPacketListHead,
   UInt32 inMaxChunkDataSize,
   SHChunkRecord *inChunk,
   SInt32 inFlags
);
```

Parameters

rtpr

The component instance of your packet reassembler.

inPacketListHead

A pointer to the list of packets that make up this chunk.

inMaxChunkDataSize

An unsigned 32-bit integer containing the maximum allowable chunks size.

inChunk

A pointer to the chunk record. Write the chunk data to this record.

inFlags

A 32-bit signed integer containing any flags being passed to your media packetizer.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function is useful, for example, when an H.261 packet reassembler must adjust the byte at packet boundaries. Implement this function only if you need to override the base reassembler's default behavior. If you do not implement this function, the base reassembler will write the chunk data by taking `dataLength` bytes from each packet, starting at an offset of (`packetData` + `transportHeaderLength` + `payloadHeaderLength`).

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QTStreamingComponents.h`

RTPRssmDecrChunkRefCount

Tells the base reassembler to dispose of a chunk that it has created or preserved for you.

```
ComponentResult RTPRssmDecrChunkRefCount (
   RTPReassembler rtpr,
   SHChunkRecord *inChunk
);
```

Parameters

rtpr

The component instance of the base reassembler component.

inChunk

A pointer to the chunk record to dispose.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
If you have overridden `RTPRssmSendPacketList` (page 1924) behavior, and are instructing the base reassembler to construct chunks manually, your packet assembler must explicitly dispose of the chunks by calling either this function or `RTPRssmSendChunkAndDecrRefCount` (page 1923). This function is also used to release a chunk you have preserved using `RTPRssmIncrChunkRefCount` (page 1919).

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTSPketizerReassem
QTSPketizerReassem.win
qtstreaming
qtstreaming.win

Declared In
`QTStreamingComponents.h`

RTPRssmFillPacketListParams

Fills in a packet structure manually.

```
ComponentResult RTPRssmFillPacketListParams (
   RTPReassembler rtpr,
   RTPRssmPacket *inPacketListHead,
   SInt32 inNumWraparounds,
   SInt32 inFlags
);
```

Parameters

rtpr
The component instance of the base reassembler.

inPacketListHead
A pointer to the `RTPRssmPacket` packet structure.

inNumWraparounds
The high-order 32 bits of the timestamp for this packet. The low-order 32 bits are found in the RTP packet header.

inFlags
A signed 32-bit integer containing any flags you are passing to the base reassembler.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Call this function only if your packet reassembler is overriding the `RTPRssmSendPacketList` (page 1924) behavior. The base reassembler will call back to your packet reassembler using `RTPRssmAdjustPacketParams` (page 1907) and `RTPRssmComputeChunkSize` (page 1909).

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

RTPRssmGetCapabilities

Obtains the current flag settings for the base reassembler.

```
ComponentResult RTPRssmGetCapabilities (
   RTPReassembler rtpr,
   SInt32 *outFlags
);
```

Parameters

rtpr

The component instance of the base reassembler.

outFlags

On return, contains a pointer to the reassembler's current flags (see below). See these constants:

`kRTPRssmEveryPacketAChunkFlag`

`kRTPRssmQueueAndUseMarkerBitFlag`

`kRTPRssmTrackLostPacketsFlag`

`kRTPRssmNoReorderingRequiredFlag`

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Your packet reassembler can call this function at any time.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

RTPRssmGetChunkAndIncrRefCount

Causes the base reassembler to create a chunk for you manually.

```
ComponentResult RTPRssmGetChunkAndIncrRefCount (
   RTPReassembler rtpr,
   UInt32 inChunkDataSize,
   const TimeValue64 *inChunkPresentationTime,
   SHChunkRecord **outChunk
);
```

Parameters

rtpr

The component instance of the base reassembler component.

inChunkDataSize

An unsigned 32-bit integer containing the size of the chunk's data portion, in bytes.

inChunkPresentationTime

A pointer to a 64-bit time value specifying the time at which this chunk should be presented, in units of the stream's time scale.

outChunk

On return, contains a pointer to a newly-created `SHChunkRecord` structure.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function is useful if you are overriding the `RTPRssmSendPacketList` (page 1924) behavior and constructing the chunk yourself. You must explicitly dispose of the chunk when you are done with it by calling either `RTPRssmDecrChunkRefCount` (page 1910) or `RTPRssmSendChunkAndDecrRefCount` (page 1923).

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QTStreamingComponents.h`

RTPRssmGetExtChunkAndIncrRefCount

Undocumented

```
ComponentResult RTPRssmGetExtChunkAndIncrRefCount (
   RTPReassembler rtpr,
   UInt32 inChunkDataSize,
   const TimeValue64 *inChunkPresentationTime,
   SInt32 inFlags,
   SHExtendedChunkRecord **outChunk
);
```

Parameters

rtpr

Undocumented

inChunkDataSize

Undocumented

inChunkPresentationTime

 Undocumented

inFlags

 Undocumented

outChunk

 A pointer to a pointer to a `SHExtendedChunkRecord` data structure.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 6. Can be used only with Mac OS X 10.1 and later.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`QTStreamingComponents.h`

RTPRssmGetInfo

Obtains information about your packet reassembler.

```
ComponentResult RTPRssmGetInfo (
   RTPReassembler rtpr,
   OSType inSelector,
   void *ioParams
);
```

Parameters

rtpr

 The component instance of your packet reassembler.

inSelector

 A selector (see below) for the information desired. See these constants:

 `kQTSSourceTrackIDInfo`

 `kQTSSourceLayerInfo`

 `kQTSSourceLanguageInfo`

 `kQTSSourceTrackFlagsInfo`

 `kQTSSourceDimensionsInfo`

 `kQTSSourceVolumesInfo`

 `kQTSSourceMatrixInfo`

 `kQTSSourceClipRectInfo`

 `kQTSSourceGraphicsModeInfo`

 `kQTSSourceScaleInfo`

 `kQTSSourceBoundingRectInfo`

 `kQTSSourceUserDataInfo`

 `kQTSSourceInputMapInfo`

ioParams
A pointer to a data structure appropriate for the type of data requested (see below) . If your component understands the selector, write the requested information into the data structure this parameter points to.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Implement this function only for the selectors you understand. Delegate this function to the base reassembler for any other selectors. The base reassembler will correctly return an error if it doesn't understand the selector either.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTSPketizerReassem

QTSPketizerReassem.win

qtstreaming

qtstreaming.win

Declared In
`QTStreamingComponents.h`

RTPRssmGetPayloadHeaderLength

Obtains the current value of the fixed payload header length from the base reassembler.

```
ComponentResult RTPRssmGetPayloadHeaderLength (
   RTPReassembler rtpr,
   UInt32 *outPayloadHeaderLength
);
```

Parameters

rtpr
The component instance of the base reassembler component.

outPayloadHeaderLength
On return, contains a pointer to an unsigned 32-bit integer containing the length of the payload header in bytes. If your packet reassembler does not implement `RTPRssmAdjustPacketParams` (page 1907), or takes no action, the default `payloadHeaderLength` is the fixed header length that is set (default is 0).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Your packet reassembler can call this function at any time.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

RTPRssmGetStreamHandler

Obtains the component instance of the stream handler to which the base reassembler is sending your output.

```
ComponentResult RTPRssmGetStreamHandler (
   RTPReassembler rtpr,
   ComponentInstance *outStreamHandler
);
```

Parameters

rtpr

The component instance of the base reassembler.

outStreamHandler

On return, contains a pointer to the component instance of the stream handler your output is being sent to by the base reassembler.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

RTPRssmGetTimeScale

Obtains the current time scale from the base reassembler.

```
ComponentResult RTPRssmGetTimeScale (
   RTPReassembler rtpr,
   TimeScale *outSHTimeScale
);
```

Parameters

rtpr

The component instance of the base reassembler.

outSHTimeScale

On return, contains a pointer to the time scale in use by the stream handler that is processing your output.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

RTPRssmGetTimeScaleFromPacket

Lets your packet reassembler extract the time scale from a received packet and return it to the base reassembler.

```
ComponentResult RTPRssmGetTimeScaleFromPacket (
   RTPReassembler rtpr,
   QTSStreamBuffer *inStreamBuffer,
   TimeScale *outTimeScale
);
```

Parameters

rtpr
The component instance of your packet reassembler.

inStreamBuffer
A pointer to a received packet from which you may be able to extract a time scale.

outTimeScale
Return a pointer to a valid time scale or return an error. If you return a time scale, the packet will be processed normally. If you return an error, the packet will be discarded.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
If your packet reassembler has not specified a time scale as part of `RTPRssmNewStreamHandler` (page 1920), or by calling `RTPRssmSetTimeScale` (page 1930), the base reassembler calls this function when it receives packets, which allows your packet reassembler to extract the time scale from a received packet and return it to the base reassembler. Your packet reassembler must set a time scale for the stream handler before the base reassembler can process any incoming packets. If your packet reassembler doesn't know the time scale of its media in advance, because the time scale is contained in the packet header for example, the base reassembler will prompt you for a time scale whenever it receives a packet. If your packet reassembler always uses the same time scale, it should set the time scale when it opens a stream handler, and it does not need to implement this function. The base reassembler will discard received packets until it has been given a valid time scale.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

RTPRssmHandleNewPacket

Called whenever a new packet arrives, giving your packet reassembler the opportunity to process the packet.

```
ComponentResult RTPRssmHandleNewPacket (
   RTPReassembler rtpr,
   QTSStreamBuffer *inStreamBuffer,
   SInt32 inNumWraparounds
);
```

Parameters

rtpr

The component instance of your packet reassembler.

inStreamBuffer

A pointer to the newly-arrived packet.

inNumWraparounds

The upper 32 bits of the 64-bit timestamp (the lower 32 bits are in the RTP packet timestamp).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

You should implement this function only if you need to process the packet yourself, or if you need to extract information from the packets as they arrive (you need to monitor the payload header, for example). If you implement this function, you can process the packet as needed, then delegate the default processing to the base reassembler.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QTStreamingComponents.h`

RTPRssmHasCharacteristic

Determines what features your reassembler supports.

```
ComponentResult RTPRssmHasCharacteristic (
   RTPReassembler rtpr,
   OSType inCharacteristic,
   Boolean *outHasIt
);
```

Parameters

rtpr

The component instance of your packet reassembler.

inCharacteristic

A constant that defines the characteristic being tested.

outHasIt

A pointer to a Boolean value that is TRUE if your packet reassembler has the characteristic, FALSE otherwise.

Return Value
See Error Codes. Returns noErr if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTSPketizerReassem

QTSPketizerReassem.win

qtstreaming

qtstreaming.win

Declared In
QTStreamingComponents.h

RTPRssmIncrChunkRefCount

Tells the base reassembler to keep a copy of the most recent chunk after it has been sent.

```
ComponentResult RTPRssmIncrChunkRefCount (
   RTPReassembler rtpr,
   SHChunkRecord *inChunk
);
```

Parameters

rtpr

The component instance of the base reassembler.

inChunk

A pointer to the chunk record you want to preserve.

Return Value
See Error Codes. Returns noErr if there is no error.

Discussion
This function is used to assist in loss recovery, for example. You must call RTPRssmDecrChunkRefCount (page 1910) to release the chunk when you no longer need it.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTSPketizerReassem

QTSPketizerReassem.win

qtstreaming

qtstreaming.win

Declared In
QTStreamingComponents.h

RTPRssmInitialize

Called when the base reassembler is ready to have your packet reassembler begin handling media packets.

```
ComponentResult RTPRssmInitialize (
   RTPReassembler rtpr,
   RTPRssmInitParams *inInitParams
);
```

Parameters

rtpr

The component instance of your packet reassembler

inInitParams

A pointer to an `RTPRssmInitParams` structure. Use the information contained in this structure to initialize your component.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function is not called when the base reassembler opens your component for payload registration information.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

QTSPketizerReassem

QTSPketizerReassem.win

qtstreaming

qtstreaming.win

Declared In

`QTStreamingComponents.h`

RTPRssmNewStreamHandler

Opens a new stream handler and closes any currently-open stream handler.

```
ComponentResult RTPRssmNewStreamHandler (
   RTPReassembler rtpr,
   OSType inSHType,
   SampleDescriptionHandle inSampleDescription,
   TimeScale inSHTimeScale,
   ComponentInstance *outHandler
);
```

Parameters

rtpr

The component instance of the base reassembler.

inSHType

The stream handler type.

inSampleDescription

A handle to a `SampleDescription` structure appropriate for this media type. Pass in `NIL` if you don't know the `media` type yet. This structure is passed by reference; the caller is responsible for maintaining it.

inSHTimeScale

The time scale for the stream handler to use. Pass in 0 if the time scale is not yet known.

outHandler

On return, contains a pointer to the component instance of the stream handler that has been opened.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

You must pass in a valid `SampleDescription` structure and time scale before the stream handler can process packets. If you do not pass them as part of this function, do so using `RTPRssmSetTimeScale` (page 1930) and `RTPRssmSetSampleDescription` (page 1928).

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

QTSPketizerReassem

QTSPketizerReassem.win

qtstreaming

qtstreaming.win

Declared In

`QTStreamingComponents.h`

RTPRssmReleasePacketList

Releases memory associated with a packet list that your packet reassembler created itself, or a list your reassembler took ownership of as a result of implementing RTPRssmSendPacketList.

```
ComponentResult RTPRssmReleasePacketList (
   RTPReassembler rtpr,
   RTPRssmPacket *inPacketListHead
);
```

Parameters

rtpr

The component instance of the base reassembler.

inPacketListHead

A pointer to the packet list to dispose of.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This is a housekeeping function that you do not need to perform for packet lists created and handled by the base reassembler, only for packet lists that you create or take ownership of yourself.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

RTPRssmReset

Called to reset all packet reassembler and base reassembler variables for a new run of data.

```
ComponentResult RTPRssmReset (
   RTPReassembler rtpr,
   SInt32 inFlags
);
```

Parameters

`rtpr`
: The component instance of your reassembler.

`inFlags`
: A signed 32-bit integer containing any flags being passed. No flags are currently defined.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function differs from `RTPRssmClearCachedPackets` (page 1908), which disposes of the packets but still retains the last sequence number and related information; this function resets all variables as if the reassembler were just opened.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTSPketizerReassem

QTSPketizerReassem.win

qtstreaming

qtstreaming.win

Declared In
`QTStreamingComponents.h`

RTPRssmSendChunkAndDecrRefCount

Called by the packet reassembler when it has finished constructing a chunk and wants the base reassembler to send it to the stream handler.

```
ComponentResult RTPRssmSendChunkAndDecrRefCount (
   RTPReassembler rtpr,
   SHChunkRecord *inChunk,
   const SHServerEditParameters *inServerEdit
);
```

Parameters

rtpr

The component instance of the base reassembler.

inChunk

A pointer to an `SHChunkRecord` structure.

inServerEdit

A pointer to an `SHServerEditParameters` structure containing the server edit parameters. Pass in `NIL` if there is no server edit.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Use this function to manually send a chunk if you have overridden the default behavior of `RTPRssmSendPacketList` (page 1924). This function will decrement the reference count of the chunk.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QTStreamingComponents.h`

RTPRssmSendLostChunk

Allows the base reassembler to send loss notification to the stream handler.

```
ComponentResult RTPRssmSendLostChunk (
   RTPReassembler rtpr,
   const TimeValue64 *inChunkPresentationTime
);
```

Parameters

rtpr

The component instance of the base reassembler.

inChunkPresentationTime

A pointer to a 64-bit time value indicating when the chunk would have been presented, in units of the stream's time scale.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Loss notification is normally performed automatically by the base reassembler. Use this function if you are handling losses or sending chunks manually.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

RTPRssmSendPacketList

Called when the base reassembler is ready to send a sample or chunk based on a list of packets.

```
ComponentResult RTPRssmSendPacketList (
   RTPReassembler rtpr,
   RTPRssmPacket *inPacketListHead,
   const TimeValue64 *inLastChunkPresentationTime,
   SInt32 inFlags
);
```

Parameters

`rtpr`
The component instance of your packet reassembler.

`inPacketListHead`
A pointer to the packet list.

`inLastChunkPresentationTime`
A pointer to a time value which specifies when to present this chunk, in units of the stream's time scale.

`inFlags`
A signed 32-bit integer containing any flags being passed (see below). See these constants:
`kRTPRssmLostSomePackets`

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Implement this call if your packet reassembler needs to modify the packet list, or if it overrides the default handling of packet loss. If you do not implement this call, the base reassembler will adjust the `packet` parameters on all packets in the list, compute the chunk size, and send the chunk. If packet loss has occurred, all the packets will be discarded and the stream handler will be informed that the chunk has been lost.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTSPketizerReassem

QTSPketizerReassem.win
qtstreaming
qtstreaming.win

Declared In
`QTStreamingComponents.h`

RTPRssmSendStreamBufferRange

Notifies the base reassembler to construct and send a chunk based on a part of the stream buffer.

```
ComponentResult RTPRssmSendStreamBufferRange (
   RTPReassembler rtpr,
   RTPSendStreamBufferRangeParams *inParams
);
```

Parameters

rtpr
: The component instance of the base reassembler.

inParams
: A pointer to an `RTPSendStreamBufferRangeParams` structure, which specifies the stream buffer, presentation time, start position in the buffer, length of the data in bytes, and any flags.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
The contents of the stream buffer will be referenced, not copied. You are responsible for maintaining valid data in the stream buffer.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

RTPRssmSendStreamHandlerChanged

Called when you have changed something in the stream handler and you want the notification propagated.

```
ComponentResult RTPRssmSendStreamHandlerChanged (
   RTPReassembler rtpr
);
```

Parameters

rtpr
: The component instance of the base reassembler.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function is useful, for example, if you have changed the dimensions of the video.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

RTPRssmSetCapabilities

Sets the capabilities of a streaming packet reassembler.

```
ComponentResult RTPRssmSetCapabilities (
   RTPReassembler rtpr,
   SInt32 inFlags,
   SInt32 inFlagsMask
);
```

Parameters

rtpr

The component instance of the base reassembler.

inFlags

A signed 32-bit integer containing the logical OR of all the flags (see below) you are setting. See these constants:

`kRTPRssmEveryPacketAChunkFlag`

`kRTPRssmQueueAndUseMarkerBitFlag`

`kRTPRssmTrackLostPacketsFlag`

`kRTPRssmNoReorderingRequiredFlag`

inFlagsMask

Use this field to preserve the state of any flags you do not wish to alter. If a flag (see below) is set in this field, and is not set in the `inFlags` field, it will not be changed from its current setting.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Your packet reassembler can call this function at any time.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTSPketizerReassem

QTSPketizerReassem.win

qtstreaming

qtstreaming.win

Declared In
`QTStreamingComponents.h`

RTPRssmSetInfo

Sets various parameters of your packet reassembler; it is also called to set parameters of the base reassembler.

```
ComponentResult RTPRssmSetInfo (
   RTPReassembler rtpr,
   OSType inSelector,
   void *ioParams
);
```

Parameters

rtpr
The component instance of your packet reassembler.

inSelector
A selector (see below) for the information being set. Ignore any selectors you do not understand. See these constants:
`kQTSSourceTrackIDInfo`
`kQTSSourceLayerInfo`
`kQTSSourceLanguageInfo`
`kQTSSourceTrackFlagsInfo`
`kQTSSourceDimensionsInfo`
`kQTSSourceVolumesInfo`
`kQTSSourceMatrixInfo`
`kQTSSourceClipRectInfo`
`kQTSSourceGraphicsModeInfo`
`kQTSSourceScaleInfo`
`kQTSSourceBoundingRectInfo`
`kQTSSourceUserDataInfo`
`kQTSSourceInputMapInfo`

ioParams
A pointer to the information that should be set.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Delegate this function to the base reassembler for any selectors you don't understand. If the base reassembler doesn't understand them either, it will return an error to the caller.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In

`QTStreamingComponents.h`

RTPRssmSetPayloadHeaderLength

Called by the packet reassembler to set a fixed header length for your payload.

```
ComponentResult RTPRssmSetPayloadHeaderLength (
   RTPReassembler rtpr,
   UInt32 inPayloadHeaderLength
);
```

Parameters

rtpr

The component instance of the base reassembler.

inPayloadHeaderLength

An unsigned 32-bit integer containing the fixed payload header length, in bytes. If your packet reassembler does not implement `RTPRssmAdjustPacketParams` (page 1907), or takes no action, the default `payloadHeaderLength` is the fixed header length that is set (default is 0).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QTStreamingComponents.h`

RTPRssmSetSampleDescription

Changes the SampleDescription structure being used by the stream handler; all subsequent samples will be marked with this new structure.

```
ComponentResult RTPRssmSetSampleDescription (
   RTPReassembler rtpr,
   SampleDescriptionHandle inSampleDescription
);
```

Parameters

rtpr

The component instance of the base reassembler.

inSampleDescription

The handle of a `SampleDescription` structure to use. You are responsible for keeping the handle and the data structure valid during subsequent operations.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion
The `SampleDescription` structure is not passed on a per-packet basis, but a per-sample basis, so the `SampleDescription` structure should not be changed until a complete sample (sometimes called a "frame" or "chunk") has been reassembled.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTSPketizerReassem

QTSPketizerReassem.win

qtstreaming

qtstreaming.win

Declared In
`QTStreamingComponents.h`

RTPRssmSetStreamHandler

Assigns a stream handler to the output of the base reassembler.

```
ComponentResult RTPRssmSetStreamHandler (
   RTPReassembler rtpr,
   ComponentInstance inStreamHandler
);
```

Parameters

rtpr
The component instance of the base reassembler.

inStreamHandler
The component instance of the stream handler to use.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
The stream handler must already be opened and initialized, and its time scale must already be set.

Special Considerations
Use this function only if you have opened and initialized a stream handler yourself. The base reassembler will not close the stream handler it is already using.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

RTPRssmSetTimeScale

Sets the time scale for the stream handler that will render your output.

```
ComponentResult RTPRssmSetTimeScale (
   RTPReassembler rtpr,
   TimeScale inSHTimeScale
);
```

Parameters

rtpr

The component instance of the base reassembler

inSHTimeScale

The time scale for the stream handler to use

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The time scale is the number of time units that pass in one second for the media whose sample data is carried in this stream. The stream handler's time scale must be set before it can deliver any data to the user.

Special Considerations

This function is normally used by a packet reassembler when the time scale to use is not initially known. You don't need to call this function if you specified a time scale when the stream handler was opened.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QTStreamingComponents.h`

TerminateQTS

Terminates the QuickTime Streaming toolbox.

```
OSErr TerminateQTS (
   void
);
```

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeStreaming.h`

Callbacks

QTSNotificationProc

A back channel from a presentation to its creator, sending notification of various events such as a presentation, ending, or acknowledgment of a preroll request.

```
typedef ComponentResult (*QTSNotificationProcPtr) (ComponentResult inErr, OSType
inNotificationType, void *inNotificationParams, void *inRefCon);
```

If you name your function `MyQTSNotificationProc`, you would declare it this way:

```
ComponentResult MyQTSNotificationProc (
    ComponentResult    inErr,
    OSType             inNotificationType,
    void               *inNotificationParams,
    void               *inRefCon );
```

Parameters

inErr
: *Undocumented*

inNotificationType
: The kind of notification; see `QuickTimeStreaming.h`.

inNotificationParams
: *Undocumented*

inRefCon
: *Undocumented*

Return Value

See `Error Codes`. Your callback should return `noErr` if there is no error.

Declared In

`QuickTimeStreaming.h`, `QTStreamingComponents.h`

RTPMPDataReleaseProc

Routine called when a media packetizer is finished with its sample data.

```
typedef void (*RTPMPDataReleaseProcPtr) (UInt8 *inData, void *inRefCon);
```

If you name your function `MyRTPMPDataReleaseProc`, you would declare it this way:

```
void MyRTPMPDataReleaseProc (
    UInt8    *inData,
    void     *inRefCon );
```

Parameters

inData
: A pointer to the data.

inRefCon

A pointer to information passed from a `RTPMPSampleDataParams` structure.

Declared In

`QuickTimeStreaming.h, QTStreamingComponents.h`

RTPPBCallbackProc

Routine used to communicate with the caller of a media packetizer.

```
typedef void (*RTPPBCallbackProcPtr) (OSType inSelector, void *ioParams, void
*inRefCon);
```

If you name your function `MyRTPPBCallbackProc`, you would declare it this way:

```
void MyRTPPBCallbackProc (
    OSType    inSelector,
    void      *ioParams,
    void      *inRefCon );
```

Parameters

inSelector

Undocumented

ioParams

Undocumented

inRefCon

Undocumented

Declared In

`QuickTimeStreaming.h, QTStreamingComponents.h`

Data Types

MediaPacketizerRequirements

Stores the functional requirements for a media packetizer.

```
struct MediaPacketizerRequirements {
     OSType    mediaType;
     OSType    dataFormat;
     UInt32    capabilityFlags;
     UInt8     canPackMatrixType;
     UInt8     pad[3];
 };
```

Fields

`mediaType`

Discussion

Media type required; see `Data References`. 0 means all media types.

`dataFormat`

Discussion
Data format required; see `Media Identifiers`. 0 means all formats.

`capabilityFlags`

Discussion
Constants (see below) that indicate the packetizer's ability to handle non-standard track characteristics. See these constants:

    kMediaPacketizerCanPackEditRate
    kMediaPacketizerCanPackLayer
    kMediaPacketizerCanPackVolume
    kMediaPacketizerCanPackBalance
    kMediaPacketizerCanPackGraphicsMode
    kMediaPacketizerCanPackEmptyEdit

`canPackMatrixType`

Discussion
Constant (see below); the packetizer needs to pack any matrix type up to this level. Set to `identityMatrixType` for identity matrix (no translation) only. See these constants:

    identityMatrixType
    translateMatrixType
    scaleMatrixType
    scaleTranslateMatrixType
    linearMatrixType
    linearTranslateMatrixType
    perspectiveMatrixType

`pad`

Discussion
Unused.

Related Functions
`QTSFindMediaPacketizer` (page 1800)

Declared In
`QuickTimeStreaming.h, QTStreamingComponents.h`

MediaPacketizerRequirementsPtr

Represents a type used by the QuickTime Streaming API.

```
typedef MediaPacketizerRequirements * MediaPacketizerRequirementsPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

QTAtomSpec

Specifies an atom and its container.

```
struct QTAtomSpec {
     QTAtomContainer    container;
     QTAtom             atom;
 };
```

Fields
`container`

Discussion
A QT atom container.

`atom`

Discussion
A QT atom.

Declared In
`QuickTimeStreaming.h, QTStreamingComponents.h`

QTAtomSpecPtr

Represents a type used by the QuickTime Streaming API.

```
typedef QTAtomSpec * QTAtomSpecPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Movies.h`

QTSExportParams

Undocumented

```
struct QTSExportParams {
     SInt32               version;
     OSType               exportType;
     void                 *exportExtraData;
     OSType               destinationContainerType;
     void                 *destinationContainerData;
     void                 *destinationContainerExtras;
     SInt32               flagsIn;
     SInt32               flagsOut;
     QTSModalFilterUPP    filterProc;
     void                 *filterProcRefCon;
     Component            exportComponent;
 };
```

Fields
`version`

Discussion
Undocumented

`exportType`

Discussion
Undocumented

`exportExtraData`

Discussion
Undocumented

`destinationContainerType`

Discussion
Undocumented

`destinationContainerData`

Discussion
Undocumented

`destinationContainerExtras`

Discussion
Undocumented

`flagsIn`

Discussion
Undocumented

`flagsOut`

Discussion
Undocumented

`filterProc`

Discussion
Undocumented

`filterProcRefCon`

Discussion
Undocumented

`exportComponent`

Discussion
Undocumented

Related Functions
`QTSPresExport` (page 1829)

Declared In
`QuickTimeStreaming.h, QTStreamingComponents.h`

QTSInstantOnPref

Contains instant on information for QuickTime Streaming.

```
struct QTSInstantOnPref {
     SInt32    flags;
     SInt32    factor;
 };
```

Fields

`flags`

Discussion

Constants (see below) that enable instant on. See these constants:

  `kQTSInstantOnFlag_Enable`

  `kQTSInstantOnFlag_Permitted`

`factor`

Discussion

Values can range from 0 to 100; the default value is 50.

Version Notes

Introduced in QuickTime 6.

Related Functions

`QTSPrefsGetInstantOnSettings` (page 1827)
`QTSPrefsSetInstantOnSettings` (page 1827)

Declared In

`QuickTimeStreaming.h, QTStreamingComponents.h`

QTSMediaParams

Combines the QTSVideoParams and QTSAudioParams structures.

```
struct QTSMediaParams {
     QTSVideoParams    v;
     QTSAudioParams    a;
 };
```

Fields

`v`

Discussion

A `QTSVideoParams` structure.

`a`

Discussion

A `QTSAudioParams` structure.

Declared In

`QuickTimeStreaming.h, QTStreamingComponents.h`

QTSMemPtr

Represents a type used by the QuickTime Streaming API.

```
typedef struct OpaqueQTSMemPtr * QTSMemPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSNewPresentationParams

Specifies a presentation for QTSNewPresentation.

```
struct QTSNewPresentationParams {
     OSType                 dataType;
     const void *           data;
     UInt32                 dataLength;
     QTSEditListHandle      editList;
     SInt32                 flags;
     TimeScale              timeScale;
     QTSMediaParams *       mediaParams;
     QTSNotificationUPP     notificationProc;
     void *                 notificationRefCon;
 };
```

Fields
`dataType`

Discussion
Undocumented

`data`

Discussion
Undocumented

`dataLength`

Discussion
Undocumented

`editList`

Discussion
A handle to a `QTSEditList` structure.

`flags`

Discussion
Undocumented

`timeScale`

Discussion
The time scale; set to 0 for the default time scale.

`mediaParams`

Discussion
Undocumented

`notificationProc`

Discussion
A pointer to a `QTSNotificationProc` callback.

`notificationRefCon`

Discussion
A reference constant to be passed to the `QTSNotificationProc` callback.

Declared In
`QuickTimeStreaming.h, QTStreamingComponents.h`

QTSNoProxyPref

Provides data for the QTSPrefsGetNoProxyURLs function.

```
struct QTSNoProxyPref {
     UInt32    flags;
     UInt32    seed;
     char      urlList[1];
 };
```

Fields

`flags`

Discussion
Undocumented

`seed`

Discussion
A seed value from the last time this setting was read from the system preferences.

`urlList`

Discussion
A null-terminated, comma-delimited list of URLs.

Related Functions
`QTSPrefsGetNoProxyURLs` (page 1827)

Declared In
`QuickTimeStreaming.h, QTStreamingComponents.h`

QTSNotificationUPP

Represents a type used by the QuickTime Streaming API.

```
typedef STACK_UPP_TYPE(QTSNotificationProcPtr) QTSNotificationUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSPresentation

Represents a type used by the QuickTime Streaming API.

```
typedef QTSPresentationRecord * QTSPresentation;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeStreaming.h`

QTSPresentationRecord

Defines a presentation.

```
struct QTSPresentationRecord {
     long    data[1];
 };
```

Fields
`data`
Discussion
Array of data that constitutes the presentation.

Declared In
`QuickTimeStreaming.h`, `QTStreamingComponents.h`

QTSPresIdleParams

Provides parameters for QTSPresIdle.

```
struct QTSPresIdleParams {
     QTSStream      stream;
     TimeValue64    movieTimeToDisplay;
     SInt32         flagsIn;
     SInt32         flagsOut;
 };
```

Fields
`stream`
Discussion
A pointer to a `QTSStreamRecord` structure.

`movieTimeToDisplay`
Discussion
Undocumented

`flagsIn`
Discussion
Undocumented

`flagsOut`

Discussion
Undocumented

Related Functions
`QTSPresIdle` (page 1846)

Declared In
`QuickTimeStreaming.h, QTStreamingComponents.h`

QTSPresParams

Undocumented.

```
struct QTSPresParams {
     UInt32                 version;
     QTSEditListHandle      editList;
     SInt32                 flags;
     TimeScale              timeScale;
     QTSMediaParams         *mediaParams;
     QTSNotificationUPP     notificationProc;
     void                   *notificationRefCon;
 };
```

Fields
`version`

Discussion
Undocumented

`editList`

Discussion
Undocumented

`flags`

Discussion
Undocumented

`timeScale`

Discussion
Undocumented

`mediaParams`

Discussion
Undocumented

`notificationProc`

Discussion
Undocumented

`notificationRefCon`

Discussion
Undocumented

Related Functions
QTSNewPresentationFromData (page 1816)
QTSNewPresentationFromDataRef (page 1817)
QTSNewPresentationFromFile (page 1817)

Declared In
QuickTimeStreaming.h, QTStreamingComponents.h

QTSProxyPref

Provides data for the QTSPrefsFindProxyByType function.

```
struct QTSProxyPref {
     UInt32    flags;
     SInt32    portID;
     UInt32    seed;
     Str255    serverNameStr;
 };
```

Fields
flags

Discussion
Undocumented

portID

Discussion
ID of the port to use for this connection type.

seed

Discussion
A seed value from the last time this setting was read from the system preferences.

serverNameStr

Discussion
A proxy server URL.

Related Functions
QTSPrefsFindProxyByType (page 1824)

Declared In
QuickTimeStreaming.h, QTStreamingComponents.h

QTSSourcer

Represents a type used by the QuickTime Streaming API.

```
typedef ComponentInstance QTSSourcer;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
QTStreamingComponents.h

QTSSourcerInitParams

Holds information for initializing a streaming sourcer.

```
struct QTSSourcerInitParams {
     SInt32    version;
     SInt32    flags;
     OSType    dataType;
     void      *data;
     UInt32    dataLength;
 };
```

Fields

`version`

Discussion

Undocumented

`flags`

Discussion

Undocumented

`dataType`

Discussion

Undocumented

`data`

Discussion

Undocumented

`dataLength`

Discussion

Undocumented

Related Functions

`QTSNewSourcer` (page 1819)
`QTSSourcerInitialize` (page 1867)

Declared In

`QuickTimeStreaming.h, QTStreamingComponents.h`

QTSStatHelper

Represents a type used by the QuickTime Streaming API.

```
typedef QTSStatHelperRecord * QTSStatHelper;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeStreaming.h`

QTSStatHelperNextParams

Holds information about the next streaming statistic obtained by QTSStatHelperNext.

```
struct QTSStatHelperNextParams {
     SInt32       flags;
     OSType       returnedStatisticsType;
     QTSStream    returnedStream;
     UInt32       maxStatNameLength;
     char *       returnedStatName;
     UInt32       maxStatStringLength;
     char *       returnedStatString;
     UInt32       maxStatUnitLength;
     char *       returnedStatUnit;
 };
```

Fields

`flags`

Discussion

Undocumented See these constants:

`kQTSStatHelperReturnPascalStringsFlag`

`returnedStatisticsType`

Discussion

Undocumented

`returnedStream`

Discussion

On return, a pointer to a `QTSStreamRecord` structure.

`maxStatNameLength`

Discussion

Undocumented

`returnedStatName`

Discussion

Undocumented; pass `NIL` if you don't want this information.

`maxStatStringLength`

Discussion

Undocumented

`returnedStatString`

Discussion

Undocumented; pass `NIL` if you don't want this information.

`maxStatUnitLength`

Discussion

Undocumented

`returnedStatUnit`

Discussion

Undocumented; pass `NIL` if you don't want this information.

Discussion

When you call QTSStatHelperNext (page 1871), specifying a statistic helper and the address of this structure, QuickTime fills in this structure with information about the next statistic obtained by the statistic helper.

Related Functions

QTSStatHelperNext (page 1871)

Declared In

QuickTimeStreaming.h, QTStreamingComponents.h

QTSStatHelperRecord

Defines the component instance of a statistics helper.

```
struct QTSStatHelperRecord {
     long    data[1];
 };
```

Fields

data

Discussion

The component instance of the statistics helper.

Declared In

QuickTimeStreaming.h, QTStreamingComponents.h

QTSStream

Represents a type used by the QuickTime Streaming API.

```
typedef QTSStreamRecord * QTSStream;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

QuickTimeStreaming.h

QTSStreamBuffer

Defines a stream buffer for QuickTime streaming.

```
struct QTSStreamBuffer {
     struct QTSStreamBuffer *   reserved1;
     struct QTSStreamBuffer *   reserved2;
     struct QTSStreamBuffer *   next;
     unsigned char *            rptr;
     unsigned char *            wptr;
     long                       reserved3;
     UInt32                     metadata[4];
     SInt32                     flags;
 };
```

Fields

`reserved1`

Discussion
Reserved; do not use.

`reserved2`

Discussion
Reserved; do not use.

`next`

Discussion
A pointer to the next message block in a message.

`rptr`

Discussion
A pointer to the first byte in the data buffer that contains real data.

`wptr`

Discussion
A pointer to the byte after the last byte in the data buffer that contains real data.

`reserved3`

Discussion
Reserved; do not use.

`metadata`

Discussion
Usage defined by message sender.

`flags`

Discussion
Reserved; do not use.

Related Functions
`QTSCopyMessage` (page 1797)
`QTSDupMessage` (page 1800)
`QTSFlattenMessage` (page 1804)
`QTSFreeMessage` (page 1805)
`QTSMessageLength` (page 1814)
`RTPRssmGetTimeScaleFromPacket` (page 1917)
`RTPRssmHandleNewPacket` (page 1918)

Declared In
`QuickTimeStreaming.h, QTStreamingComponents.h`

QTSStreamRecord

Contains a stream for QuickTime streaming.

```
struct QTSStreamRecord {
     long    data[1];
 };
```

Fields

`data`

Discussion

An array of data representing the stream.

Declared In

`QuickTimeStreaming.h, QTStreamingComponents.h`

QTSTransportPref

Records streaming transport preferences.

```
struct QTSTransportPref {
     OSType    protocol;
     SInt32    portID;
     UInt32    flags;
     UInt32    seed;
 };
```

Fields

`protocol`

Discussion

Constant that identifies the streaming transport protocol; see `Streaming Transport Atoms`.

`portID`

Discussion

ID of the port to use for this connection type.

`flags`

Discussion

Connection flags (see below). See these constants:

`kConnectionActive`

`kConnectionUseSystemPref`

`seed`

Discussion

A seed value from the last time this setting was read from the system preferences.

Related Functions

`QTSPrefsFindConnectionByType` (page 1823)
`QTSPrefsGetActiveConnection` (page 1826)

Declared In

`QuickTimeStreaming.h, QTStreamingComponents.h`

RTPMediaPacketizer

Represents a type used by the QuickTime Streaming API.

```
typedef ComponentInstance RTPMediaPacketizer;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

RTPMPDataReleaseUPP

Represents a type used by the QuickTime Streaming API.

```
typedef STACK_UPP_TYPE(RTPMPDataReleaseProcPtr) RTPMPDataReleaseUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

RTPMPSampleDataParams

Holds media packetizer sample data, including any number of samples or a partial sample.

```
struct RTPMPSampleDataParams {
     UInt32                 version;
     UInt32                 timeStamp;
     UInt32                 duration;
     UInt32                 playOffset;
     Fixed                  playRate;
     SInt32                 flags;
     UInt32                 sampleDescSeed;
     Handle                 sampleDescription;
     RTPMPSampleRef         sampleRef;
     UInt32                 dataLength;
     const UInt8 *          data;
     RTPMPDataReleaseUPP    releaseProc;
     void *                 refCon;
 };
```

Fields
`version`
Discussion
Version of the data structure. Currently always 0.

`timeStamp`
Discussion
RTP time stamp for the presentation of the sample data. This time stamp has already been adjusted by edits, edit rates, etc.

`duration`

Discussion
`Duration` (in RTP time scale) of the sample. For unknown duration, enter 0.

`playOffset`

Discussion
Offset within the media sample itself. This is only used for media formats where a single media sample can span across multiple time units. QuickTime Music is an example of this, where a single sample spans the entire track. For most video and audio formats, this will be 0.

`playRate`

Discussion
1.0 (0x00010000) is normal. Higher numbers indicate faster play rates. Note that timeStamp is already adjusted by the rate. This field is generally of interest only to audio packetizers.

`flags`

Discussion
Flag (see below) to indicate if the sample is a sync sample (key frame). See these constants:
    `kRTPMPSyncSampleFlag`

`sampleDescSeed`

Discussion
If the sample description changes, this number will change.

`sampleDescription`

Discussion
The sample description for the given media sample.

`sampleRef`

Discussion
Reserved; do not use.

`dataLength`

Discussion
`Size` of the media data.

`data`

Discussion
Pointer to the media data.

`releaseProc`

Discussion
If not `NIL`, you need to call your `RTPMPDataReleaseProc` when you are finished with the sample data.

`refCon`

Discussion
Information to pass to the `RTPMPDataReleaseProc`.

Related Functions
`RTPPBAddPacketSampleData` (page 1893)

Declared In
`QuickTimeStreaming.h, QTStreamingComponents.h`

RTPPacketBuilder

Represents a type used by the QuickTime Streaming API.

```
typedef ComponentInstance RTPPacketBuilder;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

RTPPacketGroupRef

Represents a type used by the QuickTime Streaming API.

```
typedef struct OpaqueRTPPacketGroupRef * RTPPacketGroupRef;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

RTPPacketRef

Represents a type used by the QuickTime Streaming API.

```
typedef struct OpaqueRTPPacketRef * RTPPacketRef;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

RTPPacketRepeatedDataRef

Represents a type used by the QuickTime Streaming API.

```
typedef struct OpaqueRTPPacketRepeatedDataRef * RTPPacketRepeatedDataRef;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

RTPPayloadSortRequest

Specifies the sort order for a list of packetizers.

```
struct RTPPayloadSortRequest {
     long                         characteristicCount;
     RTPPayloadCharacteristic     characteristic[1];
 };
```

Fields
`characteristicCount`

Discussion
The number of structures in the characteristic field.

`characteristic`

Discussion
An array of `RTPPayloadCharacteristic` structures.

Related Functions
`QTSFindMediaPacketizer` (page 1800)
`QTSFindMediaPacketizerForPayloadID` (page 1801)
`QTSFindMediaPacketizerForPayloadName` (page 1801)
`QTSFindMediaPacketizerForTrack` (page 1802)
`QTSFindReassemblerForPayloadID` (page 1803)
`QTSFindReassemblerForPayloadName` (page 1803)

Declared In
`QuickTimeStreaming.h, QTStreamingComponents.h`

RTPPayloadSortRequestPtr

Represents a type used by the QuickTime Streaming API.

```
typedef RTPPayloadSortRequest * RTPPayloadSortRequestPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

RTPPBCallbackUPP

Represents a type used by the QuickTime Streaming API.

```
typedef STACK_UPP_TYPE(RTPPBCallbackProcPtr) RTPPBCallbackUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

RTPReassembler

Represents a type used by the QuickTime Streaming API.

```
typedef ComponentInstance RTPReassembler;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QTStreamingComponents.h`

RTPRssmInitParams

Initializes a packet reassembler component.

```
struct RTPRssmInitParams {
     RTPSSRC       ssrc;
     UInt8         payloadType;
     UInt8         pad[3];
     TimeBase      timeBase;
     TimeScale     timeScale;
 };
```

Fields
`ssrc`

Discussion
Undocumented

`payloadType`

Discussion
Undocumented

`pad`

Discussion
Unused.

`timeBase`

Discussion
A reference to the reassembler's time base. You obtain a time base by calling `GetMovieTimeBase` (page 224) or `NewTimeBase` (page 261).

`timeScale`

Discussion
The reassembler's time scale.

Related Functions
`RTPRssmInitialize` (page 1920)

Declared In
`QuickTimeStreaming.h, QTStreamingComponents.h`

RTPRssmPacket

A streaming reassembler packet list.

```
struct RTPRssmPacket {
     struct RTPRssmPacket *    next;
     struct RTPRssmPacket *    prev;
     QTSStreamBuffer *         streamBuffer;
     Boolean                   paramsFilledIn;
     UInt8                     pad[1];
     UInt16                    sequenceNum;
     UInt32                    transportHeaderLength;
     UInt32                    payloadHeaderLength;
     UInt32                    dataLength;
     SHServerEditParameters    serverEditParams;
     TimeValue64               timeStamp;
     SInt32                    chunkFlags;
     SInt32                    flags;
 };
```

Fields

`next`

Discussion
A pointer to the next `RTPRssmPacket` structure.

`prev`

Discussion
A pointer to the previous `RTPRssmPacket` structure.

`streamBuffer`

Discussion
A pointer to a `QTSStreamBuffer` structure defining the stream buffer.

`paramsFilledIn`

Discussion
Undocumented

`pad`

Discussion
Undocumented

`sequenceNum`

Discussion
Undocumented

`transportHeaderLength`

Discussion
Undocumented

`payloadHeaderLength`

Discussion
Undocumented

`dataLength`

Discussion
Undocumented

`serverEditParams`

Discussion
Undocumented

`timeStamp`

Discussion
Undocumented

`chunkFlags`

Discussion
Undocumented

`flags`

Discussion
Undocumented

Related Functions
`RTPRssmAdjustPacketParams` (page 1907)
`RTPRssmComputeChunkSize` (page 1909)
`RTPRssmCopyDataToChunk` (page 1909)
`RTPRssmFillPacketListParams` (page 1911)
`RTPRssmReleasePacketList` (page 1921)
`RTPRssmSendPacketList` (page 1924)

Declared In
`QuickTimeStreaming.h, QTStreamingComponents.h`

RTPSendStreamBufferRangeParams

Undocumented

```
struct RTPSendStreamBufferRangeParams {
     QTSStreamBuffer *                streamBuffer;
     TimeValue64                      presentationTime;
     UInt32                           chunkStartPosition;
     UInt32                           numDataBytes;
     SInt32                           chunkFlags;
     SInt32                           flags;
     const SHServerEditParameters *   serverEditParams;
 };
```

Fields
`streamBuffer`

Discussion
Undocumented

`presentationTime`

Discussion
Undocumented

`chunkStartPosition`

Discussion
Undocumented

`numDataBytes`

Discussion

Undocumented

`chunkFlags`

Discussion

Undocumented

`flags`

Discussion

Undocumented

`serverEditParams`

Discussion

Undocumented

Related Functions

`RTPRssmSendStreamBufferRange` (page 1925)

Declared In

`QuickTimeStreaming.h, QTStreamingComponents.h`

SHChunkRecord

Defines a chunk for a reassembler.

```
struct SHChunkRecord {
     UInt32                              version;
     long                                reserved1;
     SInt32                              flags;
     UInt32                              dataSize;
     const UInt8 *                       dataPtr;
     long                                reserved2;
     long                                reserved3;
     TimeValue64                         presentationTime;
     long                                reserved4;
     long                                reserved5;
     const SHServerEditParameters *      serverEditParameters;
     long                                reserved6;
     long                                reserved7;
 };
```

Fields

`version`

Discussion

Undocumented

`reserved1`

Discussion

Reserved; do not use.

`flags`

Discussion

Undocumented

`dataSize`

Discussion
The size of the chunk data.

`dataPtr`

Discussion
A pointer to the chunk data.

`reserved2`

Discussion
Reserved; do not use.

`reserved3`

Discussion
Reserved; do not use.

`presentationTime`

Discussion
Undocumented

`reserved4`

Discussion
Reserved; do not use.

`reserved5`

Discussion
Reserved; do not use.

`serverEditParameters`

Discussion
A pointer to an `SHServerEditParameters` structure containing the server edit parameters

`reserved6`

Discussion
Reserved; do not use.

`reserved7`

Discussion
Reserved; do not use.

Related Functions
`RTPRssmCopyDataToChunk` (page 1909)
`RTPRssmDecrChunkRefCount` (page 1910)
`RTPRssmGetChunkAndIncrRefCount` (page 1912)
`RTPRssmIncrChunkRefCount` (page 1919)
`RTPRssmSendChunkAndDecrRefCount` (page 1923)

Declared In
`QuickTimeStreaming.h, QTStreamingComponents.h`

SHExtendedChunkRecord

Extends an SHChunkRecord data structure.

```
struct SHExtendedChunkRecord {
     SHChunkRecord    chunk;
     SInt32           extendedFlags;
     SInt32           extendedData[10];
 };
```

Fields

`chunk`

Discussion

A `SHChunkRecord` data structure.

`extendedFlags`

Discussion

Constants (see below) that indicate what data is being added. See these constants:

`kSHExtendedChunkFlag_HasSampleCount`

`kSHExtendedChunkFlag_HasFrameLengths`

`extendedData`

Discussion

The additional data.

Version Notes

Introduced in QuickTime 6.

Related Functions

`RTPRssmGetExtChunkAndIncrRefCount` (page 1913)

Declared In

`QuickTimeStreaming.h, QTStreamingComponents.h`

SHServerEditParameters

Undocumented

```
struct SHServerEditParameters {
     UInt32         version;
     Fixed          editRate;
     TimeValue64    dataStartTime_mediaAxis;
     TimeValue64    dataEndTime_mediaAxis;
 };
```

Fields

`version`

Discussion

Undocumented

`editRate`

Discussion

Undocumented

`dataStartTime_mediaAxis`

Discussion

Undocumented

`dataEndTime_mediaAxis`

Discussion

Undocumented

Version Notes

Introduced in QuickTime 6.

Related Functions

`RTPRssmGetExtChunkAndIncrRefCount` (page 1913)

Declared In

`QuickTimeStreaming.h`, `QTStreamingComponents.h`

Constants

MediaPacketizerRequirements Values

Constants passed to MediaPacketizerRequirements.

```
enum {
  identityMatrixType             = 0x00, /* result if matrix is identity */
  translateMatrixType            = 0x01, /* result if matrix translates */
  scaleMatrixType                = 0x02, /* result if matrix scales */
  scaleTranslateMatrixType       = 0x03, /* result if matrix scales and translates
 */
  linearMatrixType               = 0x04, /* result if matrix is general 2 x 2 */
  linearTranslateMatrixType      = 0x05, /* result if matrix is general 2 x 2 and
translates */
  perspectiveMatrixType          = 0x06  /* result if matrix is general 3 x 3 */
};
enum {
  kMediaPacketizerCanPackEditRate = 1 << 0,
  kMediaPacketizerCanPackLayer  = 1 << 1,
  kMediaPacketizerCanPackVolume = 1 << 2,
  kMediaPacketizerCanPackBalance = 1 << 3,
  kMediaPacketizerCanPackGraphicsMode = 1 << 4,
  kMediaPacketizerCanPackEmptyEdit = 1 << 5
};
```

Constants

`identityMatrixType`

Matrix is identity; value is 0x00.

Available in Mac OS X v10.0 and later.

Declared in `ImageCompression.h`.

`translateMatrixType`

Matrix translates; value is 0x01.

Available in Mac OS X v10.0 and later.

Declared in `ImageCompression.h`.

`scaleMatrixType`
Matrix scales; value is 0x02.

Available in Mac OS X v10.0 and later.

Declared in `ImageCompression.h`.

`scaleTranslateMatrixType`
Matrix translates and scales; value is 0x03.

Available in Mac OS X v10.0 and later.

Declared in `ImageCompression.h`.

`linearMatrixType`
Matrix is general 2 x 2 type; value is 0x04.

Available in Mac OS X v10.0 and later.

Declared in `ImageCompression.h`.

`linearTranslateMatrixType`
Matrix is general 2 x 2 type and translates; value is 0x05

Available in Mac OS X v10.0 and later.

Declared in `ImageCompression.h`.

`perspectiveMatrixType`
Matrix is general 3 x 3 type; value is 0x06.

Available in Mac OS X v10.0 and later.

Declared in `ImageCompression.h`.

`kMediaPacketizerCanPackEditRate`
The packetizer can pack the edit rate value.

Available in Mac OS X v10.0 and later.

Declared in `QTStreamingComponents.h`.

`kMediaPacketizerCanPackLayer`
The packetizer can pack the layer number.

Available in Mac OS X v10.0 and later.

Declared in `QTStreamingComponents.h`.

`kMediaPacketizerCanPackVolume`
The packetizer can pack the sound volume value.

Available in Mac OS X v10.0 and later.

Declared in `QTStreamingComponents.h`.

`kMediaPacketizerCanPackBalance`
The packetizer can pack the sound balance value.

Available in Mac OS X v10.0 and later.

Declared in `QTStreamingComponents.h`.

`kMediaPacketizerCanPackGraphicsMode`
The packetizer can pack the graphics transfer mode value.

Available in Mac OS X v10.0 and later.

Declared in `QTStreamingComponents.h`.

Declared In
`QuickTimeStreaming.h, QTStreamingComponents.h`

QTSTransportPref Values

Constants passed to QTSTransportPref.

```
enum {
  kConnectionActive              = (1L << 0),
  kConnectionUseSystemPref       = (1L << 1)
};
```

Constants

`kConnectionActive`

The connection is active.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeStreaming.h`.

Declared In

`QuickTimeStreaming.h, QTStreamingComponents.h`

QTSStatisticsParams Values

Constants passed to QTSStatisticsParams.

```
enum {
  kQTSAllStatisticsType          = 'all ',
  kQTSShortStatisticsType        = 'shrt',
  kQTSSummaryStatisticsType      = 'summ'
};
```

Constants

`kQTSAllStatisticsType`

A full statistics helper for all statistics; constant value is `'all '`.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeStreaming.h`.

`kQTSShortStatisticsType`

A short statistics helper; constant value is `'shrt'`.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeStreaming.h`.

Declared In

`QuickTimeStreaming.h, QTStreamingComponents.h`

QTSPresGetFlags Values

Constants passed to QTSPresGetFlags.

```
enum {
  kQTSAutoModeFlag                = 0x00000001,
  kQTSDontShowStatusFlag          = 0x00000008,
  kQTSSendMediaFlag               = 0x00010000,
  kQTSReceiveMediaFlag            = 0x00020000
};
```

Declared In
`QuickTimeStreaming.h, QTStreamingComponents.h`

QTSPrefsGetActiveConnection Values

Constants passed to QTSPrefsGetActiveConnection.

```
enum {
  kQTSDirectConnectHTTPProtocol = 'http',
  kQTSDirectConnectRTSPProtocol = 'rtsp'
};
```

Declared In
`QuickTimeStreaming.h, QTStreamingComponents.h`

kQTSDontGetDataStatisticsFlag

Constants grouped with kQTSDontGetDataStatisticsFlag.

```
enum {
  kQTSGetNameStatisticsFlag     = 0x00000001,
  kQTSDontGetDataStatisticsFlag = 0x00000002,
  kQTSUpdateAtomsStatisticsFlag = 0x00000004,
  kQTSGetUnitsStatisticsFlag    = 0x00000008,
  kQTSUpdateAllIfNecessaryStatisticsFlag = 0x00010000
};
```

Constants

`kQTSGetUnitsStatisticsFlag`

The statistics helper is to get units statistics.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeStreaming.h`.

Declared In
`QuickTimeStreaming.h, QTStreamingComponents.h`

QTSPresSetInfo Values

Constants passed to QTSPresSetInfo.

```
enum {
  kQTSGetURLLink                = 'gull' /* QTSGetURLLinkRecord* */
};
```

Declared In
`QuickTimeStreaming.h, QTStreamingComponents.h`

QTSInstantOnPref Values

Constants passed to QTSInstantOnPref.

```
enum {
  kQTSInstantOnFlag_Enable      = (1L << 0), /* instant on is enabled (read/write)*/
  kQTSInstantOnFlag_Permitted    = (1L << 1) /* instant on is possible (read only)*/
};
```

Constants

`kQTSInstantOnFlag_Enable`

Instant on is enabled for read or write operations.

Available in Mac OS X v10.2 and later.

Declared in `QuickTimeStreaming.h`.

Declared In

`QuickTimeStreaming.h, QTStreamingComponents.h`

QTSMediaSetInfo Values

Constants passed to QTSMediaSetInfo.

```
enum {
  kQTSMediaPresentationInfo     = 'pres', /* QTSMediaPresentationParams* */
  kQTSMediaNotificationInfo     = 'noti', /* QTSMediaNotificationParams* */
  kQTSMediaTotalDataRateInfo    = 'dtrt', /* UInt32*, bits/sec */
  kQTSMediaLostPercentInfo      = 'lspc', /* Fixed* */
  kQTSMediaNumStreamsInfo       = 'nstr', /* UInt32* */
  kQTSMediaIndSampleDescriptionInfo = 'isdc' /* QTSMediaIndSampleDescriptionParams*
 */
};
```

Declared In

`QuickTimeStreaming.h, QTStreamingComponents.h`

QTSNewPtr Values

Constants passed to QTSNewPtr.

```
enum {
  kQTSMemAllocAllocatedInTempMem = 0x00000001,
  kQTSMemAllocAllocatedInSystemMem = 0x00000002
};
enum {
  kQTSMemAllocClearMem           = 0x00000001,
  kQTSMemAllocDontUseTempMem     = 0x00000002,
  kQTSMemAllocTryTempMemFirst    = 0x00000004,
  kQTSMemAllocDontUseSystemMem   = 0x00000008,
  kQTSMemAllocTrySystemMemFirst  = 0x00000010,
  kQTSMemAllocHoldMemory         = 0x00001000,
 kQTSMemAllocIsInterruptTime    = 0x01010000 /* currently not supported for alloc*/
};
```

Constants

`kQTSMemAllocAllocatedInSystemMem`

The block was allocated in system memory.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeStreaming.h`.

Declared In

`QuickTimeStreaming.h, QTStreamingComponents.h`

QTSSetNetworkAppName Values

Constants passed to QTSSetNetworkAppName.

```
enum {
  kQTSNetworkAppNameIsFullNameFlag = 0x00000001
};
```

Declared In

`QuickTimeStreaming.h, QTStreamingComponents.h`

QTSStatHelperNextParams Values

Constants passed to QTSStatHelperNextParams.

```
enum {
  kQTSStatHelperReturnPascalStringsFlag = 0x00000001
};
```

Declared In

`QuickTimeStreaming.h, QTStreamingComponents.h`

QTSInsertStatisticUnits Values

Constants passed to QTSInsertStatisticUnits.

```
enum {
  kQTSStatisticsNoUnitsType     = 0,
  kQTSStatisticsPercentUnitsType = 'pcnt',
  kQTSStatisticsBitsPerSecUnitsType = 'bps ',
  kQTSStatisticsFramesPerSecUnitsType = 'fps '
};
enum {
  kQTSStatisticsStreamAtomType  = 'strm',
  kQTSStatisticsNameAtomType    = 'name', /* chars only, no length or terminator
*/
  kQTSStatisticsDataFormatAtomType = 'frmt', /* OSType */
  kQTSStatisticsDataAtomType    = 'data',
  kQTSStatisticsUnitsAtomType   = 'unit', /* OSType */
  kQTSStatisticsUnitsNameAtomType = 'unin' /* chars only, no length or terminator
 */
};
```

Constants

`kQTSStatisticsFramesPerSecUnitsType`

Frames-per-second unit type; value is `'fps '`.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeStreaming.h`.

Declared In

`QuickTimeStreaming.h, QTStreamingComponents.h`

kQTSStatisticsFixedDataFormat

Constants grouped with kQTSStatisticsFixedDataFormat.

```
enum {
  kQTSStatisticsSInt32DataFormat = 'si32',
  kQTSStatisticsUInt32DataFormat = 'ui32',
  kQTSStatisticsSInt16DataFormat = 'si16',
  kQTSStatisticsUInt16DataFormat = 'ui16',
  kQTSStatisticsFixedDataFormat = 'fixd',
  kQTSStatisticsUnsignedFixedDataFormat = 'ufix',
  kQTSStatisticsStringDataFormat = 'strg',
  kQTSStatisticsOSTypeDataFormat = 'ostp',
  kQTSStatisticsRectDataFormat  = 'rect',
  kQTSStatisticsPointDataFormat = 'pont'
};
```

Constants

`kQTSStatisticsOSTypeDataFormat`

`OSType` (32-bit) format; value is `'ostp'`.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeStreaming.h`.

Declared In

`QuickTimeStreaming.h, QTStreamingComponents.h`

Streaming Transport Atoms

Identify transport atom types for QuickTime streaming.

```
enum {
  kQTSTransAndProxyAtomType     = 'strp', /* transport/proxy prefs root atom*/
  kQTSConnectionPrefsVersion    = 'vers', /*   prefs format version*/
  kQTSTransportPrefsAtomType    = 'trns', /*   tranport prefs root atom*/
  kQTSConnectionAtomType        = 'conn', /*     connection prefs atom type, one
for each transport type*/
  kQTSUDPTransportType          = 'udp ', /*     udp transport prefs*/
  kQTSHTTPTransportType         = 'http', /*     http transport prefs*/
  kQTSTCPTransportType          = 'tcp ', /*     tcp transport prefs    */
  kQTSProxyPrefsAtomType        = 'prxy', /*   proxy prefs root atom*/
  kQTSHTTPProxyPrefsType        = 'http', /*     http proxy settings*/
  kQTSRTSPProxyPrefsType        = 'rtsp', /*     rtsp proxy settings*/
  kQTSSOCKSProxyPrefsType       = 'sock', /*     socks proxy settings*/
  kQTSProxyUserInfoPrefsType    = 'user', /*   proxy username/password root atom*/
  kQTSDontProxyPrefsAtomType    = 'nopr', /*   no-proxy prefs root atom*/
  kQTSDontProxyDataType         = 'data', /*     no proxy settings*/
  kQTSInstantOnPrefsAtomType    = 'inon' /* instant on prefs*/
};
```

Declared In
`QuickTimeStreaming.h, QTStreamingComponents.h`

kRTPMPHasUserSettingsDialogCharacteristic

Constants grouped with kRTPMPHasUserSettingsDialogCharacteristic.

```
enum {
  kRTPMPNoSampleDataRequiredCharacteristic = 'nsdr',
  kRTPMPHasUserSettingsDialogCharacteristic = 'sdlg',
  kRTPMPPrefersReliableTransportCharacteristic = 'rely',
  kRTPMPRequiresOutOfBandDimensionsCharacteristic = 'robd',
  kRTPMPReadsPartialSamplesCharacteristic = 'rpsp'
};
```

Declared In
`QuickTimeStreaming.h, QTStreamingComponents.h`

kRTPInfo_FormatString

Constants grouped with kRTPInfo_FormatString.

```
enum {
  kRTPMPPayloadTypeInfo         = 'rtpp', /* RTPMPPayloadTypeParams* */
  kRTPMPRTPTimeScaleInfo        = 'rtpt', /* TimeScale* */
  kRTPMPRequiredSampleDescriptionInfo = 'sdsc', /* SampleDescriptionHandle* */
  kRTPMPMinPayloadSize          = 'mins', /* UInt32* in bytes, does not include
rtp header; default is 0 */
  kRTPMPMinPacketDuration       = 'mind', /* UInt3* in milliseconds; default is no
 min required */
  kRTPMPSuggestedRepeatPktCountInfo = 'srpc', /* UInt32* */
  kRTPMPSuggestedRepeatPktSpacingInfo = 'srps', /* UInt32* in milliseconds */
  kRTPMPMaxPartialSampleSizeInfo = 'mpss', /* UInt32* in bytes */
  kRTPMPPreferredBufferDelayInfo = 'prbd', /* UInt32* in milliseconds */
  kRTPMPPayloadNameInfo         = 'name', /* StringPtr */
  kRTPInfo_FormatString         = 'fmtp' /* char **, caller allocates ptr, callee
 disposes */
};
```

Declared In
QuickTimeStreaming.h, QTStreamingComponents.h

RTPMPInitialize Values

Constants passed to RTPMPInitialize.

```
enum {
  kRTPMPRealtimeModeFlag        = 0x00000001
};
```

Declared In
QuickTimeStreaming.h, QTStreamingComponents.h

RTPMPIdle Values

Constants passed to RTPMPIdle.

```
enum {
  kRTPMPStillProcessingData     = 0x00000001 /* not done with data you've got*/
};
```

Declared In
QuickTimeStreaming.h, QTStreamingComponents.h

kRTPMPRespectDurationFlag

Constants grouped with kRTPMPRespectDurationFlag.

```
enum {
  kRTPMPSyncSampleFlag           = 0x00000001,
  kRTPMPRespectDurationFlag      = 0x00000002
};
```

Constants

`kRTPMPSyncSampleFlag`
  The sample is a sync sample.
  Available in Mac OS X v10.0 and later.
  Declared in `QTStreamingComponents.h`.

Declared In

`QuickTimeStreaming.h, QTStreamingComponents.h`

RTPRssmSetCapabilities Values

Constants passed to RTPRssmSetCapabilities.

```
enum {
  kRTPRssmEveryPacketAChunkFlag = 0x00000001,
  kRTPRssmQueueAndUseMarkerBitFlag = 0x00000002,
  kRTPRssmTrackLostPacketsFlag  = 0x00010000,
  kRTPRssmNoReorderingRequiredFlag = 0x00020000
};
```

Declared In

`QuickTimeStreaming.h, QTStreamingComponents.h`

RTPRssmSendPacketList Values

Constants passed to RTPRssmSendPacketList.

```
enum {
  kRTPRssmLostSomePackets        = 0x00000001
};
```

Declared In

`QuickTimeStreaming.h, QTStreamingComponents.h`

SHExtendedChunkRecord Values

Constants passed to SHExtendedChunkRecord.

```
enum {
  kSHExtendedChunkFlag_HasSampleCount = 1 << 0,
  kSHExtendedChunkFlag_HasFrameLengths = 1 << 1
};
```

Constants

`kSHExtendedChunkFlag_HasSampleCount`

Sample count data is added.

Available in Mac OS X v10.2 and later.

Declared in `QTStreamingComponents.h`.

Declared In

`QuickTimeStreaming.h, QTStreamingComponents.h`

QuickTime Virtual Reality Reference

Framework:	Frameworks/QuickTime.framework
Declared in	QuickTimeVR.h

Overview

QuickTime Virtual Reality (QTVR) is Apple's cross-platform technology for creating 360-degree panoramas and object movies. Developers can use QTVR to turn photos and computer renderings into interactive 3D views and then link these into entire 3D worlds.

Functions by Task

Accessing Image Buffers

`QTVRRefreshBackBuffer` (page 2035)
Refreshes the QTVR back buffer.

`QTVRSetBackBufferImagingProc` (page 2039)
Installs or removes a QTVR back buffer imaging procedure.

`QTVRSetPrescreenImagingCompleteProc` (page 2052)
Installs or removes a prescreen buffer imaging completion procedure.

Converting Angles and Points

`QTVRAnglesToCoord` (page 1982)
Obtains a floating-point coordinate determined by a pair of pan and tilt angles.

`QTVRColumnToPan` (page 1985)
Get the pan angle that corresponds to a column number in the object image array.

`QTVRCoordToAngles` (page 1985)
Get the pan and tilt angles of a floating-point coordinate in a panorama.

`QTVRGetAngularUnits` (page 1990)
Obtains the type of unit currently used when specifying angles.

`QTVRPanToColumn` (page 2032)
Obtains the column number in the object image array that corresponds to a pan angle.

`QTVRPtToAngles` (page 2033)
Obtains the pan and tilt angles of a point.

`QTVRRowToTilt` (page 2036)
    Obtains the tilt angle that corresponds to a row number in a QTVR object image array.

`QTVRSetAngularUnits` (page 2037)
    Sets the type of unit used when specifying QTVR angles.

`QTVRTiltToRow` (page 2061)
    Obtains the row number in the QTVR object image array that corresponds to a tilt angle.

`QTVRWrapAndConstrain` (page 2063)
    Preflights a change in the viewing or control characteristics of a QTVR object or panoramic node.

Determining Viewing Limits and Constraints

`QTVRGetConstraints` (page 1996)
    Obtains the current constraints of a QuickTime VR movie.

`QTVRGetConstraintStatus` (page 1996)
    Obtains the set of constraints active for the current view.

`QTVRGetViewingLimits` (page 2014)
    Obtains the current viewing limits of a QuickTime VR movie.

`QTVRSetConstraints` (page 2041)
    Sets the constraints of a VR movie.

Getting Scene and Node Information

`QTVRGetCurrentNodeID` (page 1999)
    Obtains the current node of a movie.

`QTVRGetNodeInfo` (page 2007)
    Obtains the node information atom container that describes a node and all the hot spots in the node.

`QTVRGetNodeType` (page 2008)
    Obtains the type of a movie node.

`QTVRGetVRWorld` (page 2019)
    Obtains the VR world atom container for a movie.

`QTVRGoToNodeID` (page 2020)
    Sets the current node of a movie.

Handling Events

`QTVRGetMouseDownTracking` (page 2006)
    Obtains the current state of mouse-down tracking.

`QTVRGetMouseOverTracking` (page 2006)
    Obtains the current state of mouse-over tracking.

`QTVRMouseDown` (page 2023)
    Handles the user's clicking the mouse button when the cursor is in a QuickTime VR movie for which mouse-down tracking is disabled.

`QTVRMouseEnter` (page 2025)
: Handles the user's moving the cursor into a QuickTime VR movie for which mouse-over tracking is disabled.

`QTVRMouseLeave` (page 2025)
: Handles the user's moving the cursor out of a QuickTime VR movie for which mouse-over tracking is disabled.

`QTVRMouseStillDown` (page 2026)
: Handles the user's holding down the mouse button while the cursor is in a QuickTime VR movie for which mouse-down tracking is disabled.

`QTVRMouseStillDownExtended` (page 2027)
: Handles the user's holding down the mouse button while the cursor is in a QuickTime VR movie for which mouse-down tracking is disabled.

`QTVRMouseUp` (page 2028)
: Handles the user's releasing the mouse button while the cursor is in a QuickTime VR movie for which mouse-down tracking is disabled.

`QTVRMouseUpExtended` (page 2029)
: Handles the user's releasing the mouse button while the cursor is in a QuickTime VR movie for which mouse-down tracking is disabled.

`QTVRMouseWithin` (page 2031)
: Handles the user's leaving the cursor in a QuickTime VR movie for which mouse-over tracking is disabled.

`QTVRSetMouseDownTracking` (page 2049)
: Sets the state of mouse-down tracking.

`QTVRSetMouseOverTracking` (page 2051)
: Sets the state of mouse-over tracking.

Intercepting QuickTime VR Manager Routines

`QTVRCallInterceptedProc` (page 1984)
: Calls an intercepted QuickTime VR function from within an intercept procedure.

`QTVRInstallInterceptProc` (page 2021)
: Installs or removes an intercept procedure for a QuickTime VR Manager function.

Managing Hot Spots

`QTVREnableHotSpot` (page 1987)
: Enables or disables one or more QTVR hot spots.

`QTVRGetHotSpotRegion` (page 2002)
: Obtains the region occupied by a hot spot.

`QTVRGetHotSpotType` (page 2003)
: Obtains the type of a QuickTime VR hot spot.

`QTVRGetVisibleHotSpots` (page 2018)
: Obtains a list of the currently visible hot spots in a QuickTime VR movie.

`QTVRPtToHotSpotID` (page 2034)
: Obtains the ID of the hot spot, if any, that lies beneath a point.

`QTVRSetMouseOverHotSpotProc` (page 2050)
Installs or removes a mouse over hot spot procedure.

`QTVRTriggerHotSpot` (page 2061)
Triggers a QTVR hot spot.

Managing Imaging Characteristics

`QTVRBeginUpdateStream` (page 1983)
Begins a stream of immediate updates to a QuickTime VR movie.

`QTVREnableTransition` (page 1988)
Enables or disables a transition effect.

`QTVREndUpdateStream` (page 1989)
Ends a stream of immediate updates to a QuickTime VR movie.

`QTVRGetImagingProperty` (page 2004)
Obtains the current value of an imaging property of a movie.

`QTVRGetVisible` (page 2018)
Obtains a movie's visibility state.

`QTVRSetImagingProperty` (page 2046)
Sets the value of an imaging property of a movie.

`QTVRSetTransitionProperty` (page 2055)
Sets the value of a transition property.

`QTVRSetVisible` (page 2059)
Sets a VR movie's visibility state.

`QTVRUpdate` (page 2063)
Forces an immediate update of a QuickTime VR movie image.

Managing Object Nodes

`QTVREnableFrameAnimation` (page 1986)
Enables or disables frame animation for an object node.

`QTVREnableViewAnimation` (page 1989)
Enables or disables view animation for an object node.

`QTVRGetAnimationSetting` (page 1991)
Obtains the current state of an animation setting for an object node.

`QTVRGetControlSetting` (page 1997)
Obtains the current state of a control setting for an object node.

`QTVRGetCurrentMouseMode` (page 1998)
Obtains the current mouse control modes.

`QTVRGetCurrentViewDuration` (page 1999)
Obtains the duration of the current view of an object node.

`QTVRGetFrameAnimation` (page 2001)
Obtains the current state of frame animation for an object node.

`QTVRGetFrameRate` (page 2001)
Obtains the current frame rate of an object node.

`QTVRGetViewAnimation` (page 2012)
Obtains the current state of view animation for an object node.

`QTVRGetViewCurrentTime` (page 2013)
Obtains the current time in the current view.

`QTVRGetViewRate` (page 2016)
Obtains the current view rate of an object node.

`QTVRGetViewState` (page 2016)
Obtains the value of a view state.

`QTVRGetViewStateCount` (page 2017)
Obtains the number of view states of an object node.

`QTVRSetAnimationSetting` (page 2038)
Sets the state of an animation setting for an object node.

`QTVRSetControlSetting` (page 2042)
Sets the state of a control setting for a QTVR object node.

`QTVRSetFrameRate` (page 2045)
Sets the frame rate of an object node.

`QTVRSetViewCurrentTime` (page 2056)
Sets the time in the current QTVR view.

`QTVRSetViewRate` (page 2058)
Sets the view rate of a QTVR object node.

`QTVRSetViewState` (page 2059)
Sets the value of a QTVR view state.

Managing QuickTime VR Movie Instances

`QTVRGetQTVRInstance` (page 2009)
Obtains an instance of a QuickTime VR movie.

`QTVRGetQTVRTrack` (page 2010)
Obtains a QTVR track contained in a QuickTime movie to use in the QTVRGetQTVRInstance call.

Managing QuickTime VR Movie Interactions

`QTVRGetInteractionProperty` (page 2005)
Obtains the value of an interaction property.

`QTVRReplaceCursor` (page 2035)
Replaces any of the standard QuickTime VR cursors with your own custom cursor.

`QTVRSetEnteringNodeProc` (page 2043)
Installs or removes a node-entering procedure.

`QTVRSetInteractionProperty` (page 2047)
Sets the value of an interaction property.

`QTVRSetLeavingNodeProc` (page 2048)
: Installs or removes a node-leaving procedure.

Managing VR Memory

`QTVRGetAvailableResolutions` (page 1992)
: Obtains the image resolutions present in the current node.

`QTVRGetBackBufferMemInfo` (page 1992)
: Obtains information about the internal back buffer that QuickTime VR maintains for caching panoramic images.

`QTVRGetBackBufferSettings` (page 1994)
: Obtains information about the resolution, pixel format, and size of the back buffer maintained internally by QuickTime VR for caching a panoramic image in a particular pixel format.

`QTVRSetBackBufferPrefs` (page 2040)
: Sets the resolution, pixel format, and size of the back buffer maintained internally by QuickTime VR for caching a panoramic image in a particular pixel format.

Manipulating Viewing Angles and Zooming

`QTVRGetFieldOfView` (page 2000)
: Obtains the vertical field of view of a QuickTime VR movie.

`QTVRGetPanAngle` (page 2008)
: Obtains the pan angle of a QuickTime VR movie.

`QTVRGetTiltAngle` (page 2011)
: Obtains the tilt angle of a QuickTime VR movie.

`QTVRGetViewCenter` (page 2013)
: Obtains the view center of a QuickTime VR movie.

`QTVRInteractionNudge` (page 2022)
: Translates the image and displays the new view or rotates the object in a particular direction and displays its new appearance.

`QTVRNudge` (page 2031)
: Turns one step in a particular direction and displays the new view.

`QTVRSetFieldOfView` (page 2044)
: Sets the vertical field of view of a QuickTime VR movie.

`QTVRSetPanAngle` (page 2051)
: Sets the pan angle of a QuickTime VR movie.

`QTVRSetTiltAngle` (page 2054)
: Sets the tilt angle of a QuickTime VR movie.

`QTVRSetViewCenter` (page 2056)
: Sets the view center of a QuickTime VR movie.

`QTVRShowDefaultView` (page 2060)
: Displays the default view of a QTVR node.

Supporting Functions

DisposeQTVRBackBufferImagingUPP (page 1975)
    Disposes of a QTVRBackBufferImagingUPP pointer.

DisposeQTVREnteringNodeUPP (page 1976)
    Disposes of a QTVREnteringNodeUPP pointer.

DisposeQTVRImagingCompleteUPP (page 1976)
    Disposes of a QTVRImagingCompleteUPP pointer.

DisposeQTVRInterceptUPP (page 1977)
    Disposes of a QTVRInterceptUPP pointer.

DisposeQTVRLeavingNodeUPP (page 1977)
    Disposes of a QTVRLeavingNodeUPP pointer.

DisposeQTVRMouseOverHotSpotUPP (page 1978)
    Disposes of a QTVRMouseOverHotSpotUPP pointer.

NewQTVRBackBufferImagingUPP (page 1978)
    Allocates a Universal Procedure Pointer for the QTVRBackBufferImagingProc callback.

NewQTVREnteringNodeUPP (page 1979)
    Allocates a Universal Procedure Pointer for the QTVREnteringNodeProc callback.

NewQTVRImagingCompleteUPP (page 1980)
    Allocates a Universal Procedure Pointer for the QTVRImagingCompleteProc callback.

NewQTVRInterceptUPP (page 1980)
    Allocates a Universal Procedure Pointer for the QTVRInterceptProc callback.

NewQTVRLeavingNodeUPP (page 1981)
    Allocates a Universal Procedure Pointer for the QTVRLeavingNodeProc callback.

NewQTVRMouseOverHotSpotUPP (page 1981)
    Allocates a Universal Procedure Pointer for the QTVRMouseOverHotSpotProc callback.

QTVRGetViewParameter (page 2015)
    Undocumented

QTVRSetViewParameter (page 2057)
    Undocumented

Functions

DisposeQTVRBackBufferImagingUPP

Disposes of a QTVRBackBufferImagingUPP pointer.

```
void DisposeQTVRBackBufferImagingUPP (
   QTVRBackBufferImagingUPP userUPP
);
```

Parameters

userUPP
    A `QTVRBackBufferImagingUPP` pointer. See `Universal Procedure Pointers`.

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
vrbackbuffer

vrbackbuffer.win

vrmovies

vrmovies.win

vrscript.win

Declared In
`QuickTimeVR.h`

DisposeQTVREnteringNodeUPP

Disposes of a QTVREnteringNodeUPP pointer.

```
void DisposeQTVREnteringNodeUPP (
   QTVREnteringNodeUPP userUPP
);
```

Parameters

userUPP
: A `QTVREnteringNodeUPP` pointer. See `Universal Procedure Pointers`.

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeVR.h`

DisposeQTVRImagingCompleteUPP

Disposes of a QTVRImagingCompleteUPP pointer.

```
void DisposeQTVRImagingCompleteUPP (
   QTVRImagingCompleteUPP userUPP
);
```

Parameters

userUPP

A `QTVRImagingCompleteUPP` pointer. See `Universal Procedure Pointers`.

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
vrscript

vrscript.win

Declared In
`QuickTimeVR.h`

DisposeQTVRInterceptUPP

Disposes of a QTVRInterceptUPP pointer.

```
void DisposeQTVRInterceptUPP (
   QTVRInterceptUPP userUPP
);
```

Parameters

userUPP

A `QTVRInterceptUPP` pointer. See `Universal Procedure Pointers`.

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeVR.h`

DisposeQTVRLeavingNodeUPP

Disposes of a QTVRLeavingNodeUPP pointer.

```
void DisposeQTVRLeavingNodeUPP (
   QTVRLeavingNodeUPP userUPP
);
```

Parameters

userUPP

A `QTVRLeavingNodeUPP` pointer. See `Universal Procedure Pointers`.

Return Value

You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes

Introduced in QuickTime 4.1.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeVR.h`

DisposeQTVRMouseOverHotSpotUPP

Disposes of a QTVRMouseOverHotSpotUPP pointer.

```
void DisposeQTVRMouseOverHotSpotUPP (
   QTVRMouseOverHotSpotUPP userUPP
);
```

Parameters

userUPP

A `QTVRMouseOverHotSpotUPP` pointer. See `Universal Procedure Pointers`.

Return Value

You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes

Introduced in QuickTime 4.1.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeVR.h`

NewQTVRBackBufferImagingUPP

Allocates a Universal Procedure Pointer for the QTVRBackBufferImagingProc callback.

```
QTVRBackBufferImagingUPP NewQTVRBackBufferImagingUPP (
   QTVRBackBufferImagingProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your application-defined function.

Return Value

A new UPP; see `Universal Procedure Pointers`.

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes

Introduced in QuickTime 4.1. Replaces `NewQTVRBackBufferImagingProc`.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

vrbackbuffer

vrbackbuffer.win

vrmovies

vrmovies.win

vrscript.win

Declared In

`QuickTimeVR.h`

NewQTVREnteringNodeUPP

Allocates a Universal Procedure Pointer for the QTVREnteringNodeProc callback.

```
QTVREnteringNodeUPP NewQTVREnteringNodeUPP (
   QTVREnteringNodeProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your application-defined function.

Return Value

A new UPP; see `Universal Procedure Pointers`.

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes

Introduced in QuickTime 4.1. Replaces `NewQTVREnteringNodeProc`.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

vrscript

vrscript.win

vrspeech

Declared In

`QuickTimeVR.h`

NewQTVRImagingCompleteUPP

Allocates a Universal Procedure Pointer for the QTVRImagingCompleteProc callback.

```
QTVRImagingCompleteUPP NewQTVRImagingCompleteUPP (
   QTVRImagingCompleteProcPtr userRoutine
);
```

Parameters

`userRoutine`

A pointer to your application-defined function.

Return Value

A new UPP; see `Universal Procedure Pointers`.

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes

Introduced in QuickTime 4.1. Replaces `NewQTVRImagingCompleteProc`.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

vrscript

vrscript.win

Declared In

`QuickTimeVR.h`

NewQTVRInterceptUPP

Allocates a Universal Procedure Pointer for the QTVRInterceptProc callback.

```
QTVRInterceptUPP NewQTVRInterceptUPP (
   QTVRInterceptProcPtr userRoutine
);
```

Parameters

`userRoutine`

A pointer to your application-defined function.

Return Value

A new UPP; see `Universal Procedure Pointers`.

Discussion
This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes
Introduced in QuickTime 4.1. Replaces `NewQTVRInterceptProc`.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
vrscript

vrscript.win

vrspeech

Declared In
`QuickTimeVR.h`

NewQTVRLeavingNodeUPP

Allocates a Universal Procedure Pointer for the QTVRLeavingNodeProc callback.

```
QTVRLeavingNodeUPP NewQTVRLeavingNodeUPP (
   QTVRLeavingNodeProcPtr userRoutine
);
```

Parameters

`userRoutine`
A pointer to your application-defined function.

Return Value
A new UPP; see `Universal Procedure Pointers`.

Discussion
This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes
Introduced in QuickTime 4.1. Replaces `NewQTVRLeavingNodeProc`.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
vrscript

vrscript.win

Declared In
`QuickTimeVR.h`

NewQTVRMouseOverHotSpotUPP

Allocates a Universal Procedure Pointer for the QTVRMouseOverHotSpotProc callback.

```
QTVRMouseOverHotSpotUPP NewQTVRMouseOverHotSpotUPP (
   QTVRMouseOverHotSpotProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your application-defined function.

Return Value

A new UPP; see `Universal Procedure Pointers`.

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes

Introduced in QuickTime 4.1. Replaces `NewQTVRMouseOverHotSpotProc`.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

vrcursors

vrcursors.win

vrscript

vrscript.win

Declared In

`QuickTimeVR.h`

QTVRAnglesToCoord

Obtains a floating-point coordinate determined by a pair of pan and tilt angles.

```
OSErr QTVRAnglesToCoord (
   QTVRInstance qtvr,
   float panAngle,
   float tiltAngle,
   QTVRFloatPoint *coord
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

panAngle

A pan angle.

tiltAngle

A tilt angle.

coord

On entry, a pointer to a `QTVRFloatPoint` structure. On return, that structure is set to the coordinate of the specified movie that corresponds to the specified pan and tilt angles.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Special Considerations

`QTVRAnglesToCoord` is valid only for panoramic nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeVR.h`

QTVRBeginUpdateStream

Begins a stream of immediate updates to a QuickTime VR movie.

```
OSErr QTVRBeginUpdateStream (
   QTVRInstance qtvr,
   QTVRImagingMode imagingMode
);
```

Parameters

`qtvr`

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

`imagingMode`

An imaging mode (see below). See these constants:

`kQTVRStatic`

`kQTVRMotion`

`kQTVRAllModes`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function configures the QuickTime VR movie specified by the `qtvr` parameter for a stream of immediate updates to its movie image. After calling `QTVRBeginUpdateStream`, you perform the updates by calling `QTVRUpdate` (page 2063). When you are finished performing the updates, call `QTVREndUpdateStream` (page 1989). Issuing a stream of image updates in this manner is significantly faster than simply calling `QTVRUpdate` repeatedly (that is, not within a begin/end update sequence). Each call to this function must be balanced by a call to `QTVREndUpdateStream`, but you can nest these calls.

Special Considerations

After you call this function and before you call `QTVREndUpdateStream` (page 1989), you must not change any of the QuickTime VR movie's imaging properties.

Calling this function locks down large blocks of memory. As a result, you should minimize the amount of time before calling `QTVREndUpdateStream`.

This function is valid only for panoramic nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
vrscript
vrscript.win

Declared In
`QuickTimeVR.h`

QTVRCallInterceptedProc

Calls an intercepted QuickTime VR function from within an intercept procedure.

```
OSErr QTVRCallInterceptedProc (
   QTVRInstance qtvr,
   QTVRInterceptRecord *qtvrMsg
);
```

Parameters

`qtvr`
: An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

`qtvrMsg`
: A pointer to a `QTVRInterceptRecord` structure that specifies the function that your procedure is intercepting and the parameters for that function. This should be the same intercept record passed to your intercept procedure.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function executes the QuickTime VR Manager function indicated by the `selector` field of the `qtvrMsg` intercept record. The parameters passed to that function are the QuickTime VR movie specified by the `qtvr` parameter and any other parameters contained in the `parameter` field of the `qtvrMsg` record. You can, if you wish, change the `parameters` in that field before calling this function.

You can call this function more than once in your intercept procedure. In addition, the QuickTime VR Manager will call the intercepted function again unless your intercept procedure returns TRUE in the `cancel` parameter.

Special Considerations

You should call `QTVRCallInterceptedProc` only in an intercept procedure.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
vrspeech

Declared In
`QuickTimeVR.h`

QTVRColumnToPan

Get the pan angle that corresponds to a column number in the object image array.

```
float QTVRColumnToPan (
   QTVRInstance qtvr,
   short column
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

column

A column number.

Return Value

The pan angle that corresponds to the zero-based column number in the object image array specified by the `column` parameter.

Special Considerations

This function is valid only for object nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeVR.h`

QTVRCoordToAngles

Get the pan and tilt angles of a floating-point coordinate in a panorama.

```
OSErr QTVRCoordToAngles (
   QTVRInstance qtvr,
   QTVRFloatPoint *coord,
   float *panAngle,
   float *tiltAngle
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

coord

On entry, a pointer to a `QTVRFloatPoint` structure that specifies a coordinate in the full panorama.

panAngle

On entry, a pointer to a floating-point value. On return, that value contains the pan angle of the specified coordinate.

tiltAngle
: On entry, a pointer to a floating-point value. On return, that value contains the tilt angle of the specified coordinate.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function returns, in the floating-point values pointed to by the `panAngle` and `tiltAngle` parameters, the pan and tilt angles of the point specified by the `coord` parameter. This function is useful for setting up angles in a back buffer imaging procedure; if you know a coordinate in the back buffer, you can call `QTVRCoordToAngles` to get the corresponding angles.

Special Considerations
`QTVRCoordToAngles` is valid only for panoramic nodes.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeVR.h`

QTVREnableFrameAnimation

Enables or disables frame animation for an object node.

```
OSErr QTVREnableFrameAnimation (
   QTVRInstance qtvr,
   Boolean enable
);
```

Parameters

qtvr
: An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

enable
: A Boolean value that indicates whether to enable (TRUE) or disable (FALSE) frame animation for the specified object node.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function enables or disables the frame animation state for the object node specified by the `qtvr` parameter, according to the value of the `enable` parameter. The current frame rate, set by the function `QTVRSetFrameRate` (page 2045), is unaffected by the state of frame animation of an object node.

Special Considerations
This function is valid only for object nodes. You should use this function instead of standard QuickTime functions to control object animation.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
vrscript

vrscript.win

Declared In
`QuickTimeVR.h`

QTVREnableHotSpot

Enables or disables one or more QTVR hot spots.

```
OSErr QTVREnableHotSpot (
   QTVRInstance qtvr,
   UInt32 enableFlag,
   UInt32 hotSpotValue,
   Boolean enable
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

enableFlag

The kind of hot spot or hot spots to enable or disable (see below). See these constants:

`kQTVRHotSpotID`

`kQTVRHotSpotType`

`kQTVRAllHotSpots`

hotSpotValue

The desired hot spot or spots, defined by the specified enabled flag (see below).

enable

A Boolean value that indicates whether the specified hot spots are to be enabled (TRUE) or disabled (FALSE).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function either enables or disables the hot spot or spots specified by the `enableFlag` and `hotSpotValue` parameters, according to the value of the `enable` parameter. The hot spots are always selected from among the hot spots in the current node of the QuickTime VR movie specified by the `qtvr` parameter.

Normally, all hot spots in a node are enabled (that is, the cursor automatically changes shape when it is moved over a hot spot, and the `QTVRTriggerHotSpot` (page 2061) function is called internally when the user clicks a hot spot). When a hot spot is disabled, QuickTime VR behaves as if the hot spot were not present.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
vrscript

vrscript.win

Declared In
`QuickTimeVR.h`

QTVREnableTransition

Enables or disables a transition effect.

```
OSErr QTVREnableTransition (
   QTVRInstance qtvr,
   UInt32 transitionType,
   Boolean enable
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

transitionType

A type of transition property (see below). Currently only one constant is available for this parameter. See these constants:

`kQTVRTransitionSwing`

enable

A Boolean value that indicates whether the specified transition property is to be enabled (TRUE) or disabled (FALSE).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function enables or disables the transition property specified by the `transitionType` parameter for the movie specified by the `qtvr` parameter, as indicated by the value of the `enable` parameter. Once a transition effect is enabled, it is used at the appropriate time until it is disabled by a subsequent call to this function.

Special Considerations

`QTVREnableTransition` is valid only for panoramic nodes.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
vrscript
vrscript.win

Declared In
`QuickTimeVR.h`

QTVREnableViewAnimation

Enables or disables view animation for an object node.

```
OSErr QTVREnableViewAnimation (
   QTVRInstance qtvr,
   Boolean enable
);
```

Parameters

qtvr
An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

enable
A Boolean value that indicates whether to enable (TRUE) or disable (FALSE) view animation for the specified object node.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
You should use this function instead of standard QuickTime functions to control object animation.

Special Considerations
This function is valid only for object nodes.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
vrscript
vrscript.win

Declared In
`QuickTimeVR.h`

QTVREndUpdateStream

Ends a stream of immediate updates to a QuickTime VR movie.

```
OSErr QTVREndUpdateStream (
   QTVRInstance qtvr
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function unlocks the memory locked by the matching call to `QTVRBeginUpdateStream` (page 1983) for the QuickTime VR movie specified by the `qtvr` parameter and reverses any other actions performed by that call. Each call to `QTVRBeginUpdateStream` must be balanced by a call to this function, but you can nest these calls. For nested calls, only the final call to this function unlocks the memory locked by the first call to `QTVRBeginUpdateStream`.

Special Considerations

`QTVREndUpdateStream` is valid only for panoramic nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

vrscript

vrscript.win

Declared In

`QuickTimeVR.h`

QTVRGetAngularUnits

Obtains the type of unit currently used when specifying angles.

```
QTVRAngularUnits QTVRGetAngularUnits (
   QTVRInstance qtvr
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

Return Value

The type of unit currently used (see below).

Discussion

This function returns, as its function result, a constant that indicates the type of angular unit currently used by the movie instance specified by the `qtvr` parameter. Angular values you pass to other QuickTime VR functions, such as `QTVRSetPanAngle` (page 2051), are interpreted in those units.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeVR.h`

QTVRGetAnimationSetting

Obtains the current state of an animation setting for an object node.

```
OSErr QTVRGetAnimationSetting (
   QTVRInstance qtvr,
   QTVRObjectAnimationSetting setting,
   Boolean *enable
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

setting

An animation setting (see below). See these constants:

- `kQTVRPalindromeViewFrames`
- `kQTVRDontLoopViewFrames`
- `kQTVRPlayEveryViewFrame`
- `kQTVRSyncViewToFrameRate`
- `kQTVRPalindromeViews`
- `kQTVRPlayStreamingViews`

enable

On entry, a pointer to a Boolean value. On return, that value is set to TRUE if the specified animation setting is currently enabled for the specified object node or to FALSE otherwise.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Special Considerations

This function is valid only for object nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeVR.h`

QTVRGetAvailableResolutions

Obtains the image resolutions present in the current node.

```
OSErr QTVRGetAvailableResolutions (
   QTVRInstance qtvr,
   UInt16 *resolutionsMask
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

resolutionsMask

On entry, a pointer to an unsigned short integer. On return, that integer is set to a bitmask that encodes the image resolutions (see below) available at the current node. See these constants:

`kQTVRDefaultRes`

`kQTVRFullRes`

`kQTVRHalfRes`

`kQTVRQuarterRes`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

A single node can contain multiple resolutions of a panorama or an object. The lowest order bit is always set and corresponds to the base resolution of the node. Each succeeding bit corresponds to a resolution that is half that (both horizontally and vertically) of the preceding bit. If an image with a corresponding resolution is present in the current node, that bit is set.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeVR.h`

QTVRGetBackBufferMemInfo

Obtains information about the internal back buffer that QuickTime VR maintains for caching panoramic images.

```
OSErr QTVRGetBackBufferMemInfo (
   QTVRInstance qtvr,
   UInt32 geometry,
   UInt16 resolution,
   UInt32 cachePixelFormat,
   SInt32 *minCacheBytes,
   SInt32 *suggestedCacheBytes,
   SInt32 *fullCacheBytes
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

geometry

The geometry parameter (see below) specifies the type and orientation of the panorama data. See these constants:

`kQTVRUseMovieGeometry`

`kQTVRVerticalCylinder`

resolution

The resolution for which the information is desired (see below). See these constants:

`kQTVRDefaultRes`

`kQTVRFullRes`

`kQTVRHalfRes`

`kQTVRQuarterRes`

cachePixelFormat

The desired pixel format for the back buffer. This value should be one of the defined pixel formats (see below). See these constants:

`kQTVRMinimumCache`

`kQTVRSuggestedCache`

`kQTVRFullCache`

minCacheBytes

On entry, a pointer to a long integer. On return, that long integer is set to the minimum size, in bytes, of the back buffer required to display the specified panorama with a severely limited maximum field of view. Set this parameter to `NIL` to prevent this information from being returned.

suggestedCacheBytes

On entry, a pointer to a long integer. On return, that long integer is set to the minimum size, in bytes, of the back buffer required to display the specified panorama with full wide-angle zooming. Set this parameter to `NIL` to prevent this information from being returned.

fullCacheBytes

On entry, a pointer to a long integer. On return, that long integer is set to the minimum size, in bytes, of the back buffer required to have the entire panorama in memory at once. That is the default size of the panorama back buffer. Set this parameter to `NIL` to prevent this information from being returned.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

You can use this function to get information about the size of the back buffer that would be required for caching a panoramic image of a specified pixel format, geometry, and resolution. This is a "what-if" function: you specify a resolution and a pixel format, and this function returns several buffer sizes. You can use this information, in conjunction with `QTVRSetBackBufferPrefs` (page 2040), to exercise some control over the size of the back buffer.

The resolution at which an image is to be displayed is specified by the `resolution` parameter. You can use a resolution that is not in the movie file. Relative to that resolution and the pixel depth determined by the `cachePixelFormat` parameter, this function returns, through the `minCacheBytes` parameter, the minimum size of the buffer needed to display the movie. Using a buffer of that size, however, may result in a severely limited maximum field of view. You can call the `QTVRGetViewingLimits` (page 2014) function to determine the `actual` maximum field of view.

To allow full wide-angle zooming, you should use a buffer whose size is specified by either the `suggestedCacheBytes` parameter or the `fullCacheBytes` parameter.

Special Considerations

This function is valid only for panoramic nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeVR.h`

QTVRGetBackBufferSettings

Obtains information about the resolution, pixel format, and size of the back buffer maintained internally by QuickTime VR for caching a panoramic image in a particular pixel format.

```
OSErr QTVRGetBackBufferSettings (
   QTVRInstance qtvr,
   UInt32 *geometry,
   UInt16 *resolution,
   UInt32 *cachePixelFormat,
   SInt16 *cacheSize
);
```

Parameters

`qtvr`

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

`geometry`

The type and orientation of the panorama data (see below). See these constants:

`kQTVRUseMovieGeometry`

`kQTVRVerticalCylinder`

resolution

On entry, a pointer to an unsigned short integer. On return, that integer is set to the index of the current image resolution (see below). See these constants:

`kQTVRDefaultRes`
`kQTVRFullRes`
`kQTVRHalfRes`
`kQTVRQuarterRes`

cachePixelFormat

On entry, a pointer to a long integer. On return, that long integer is set to the pixel format of the current panorama back buffer (see below). See these constants:

cacheSize

On entry, a pointer to a short integer. On return, that integer is set to a value that describes the size of the current panorama back buffer. See these constants:

`kQTVRMinimumCache`
`kQTVRSuggestedCache`
`kQTVRFullCache`

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function returns, through the `resolution` parameter, the index of the current resolution for the QuickTime VR movie specified by the `qtvr` parameter. The index indicates which bit in the mask value returned by `QTVRGetAvailableResolutions` (page 1992) specifies the current resolution. For example, if the returned index is 1, the base resolution is being used. If the returned index is 2, then a resolution of half the base resolution is being used. This function also returns the pixel format and the cache size in the `cachePixelFormat` and `cacheSize` parameters, respectively.

The QuickTime VR file might not contain an image track corresponding to the resolution indicated by the resolution value returned. The QuickTime VR Manager may have set a lower resolution because memory is low, or the resolution may have been set by a call to `QTVRSetBackBufferPrefs` (page 2040).

Special Considerations

This function is valid only for panoramic nodes.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
vrbackbuffer

vrbackbuffer.win

vrcursors

vrmakeobject

vrmovies

Declared In
`QuickTimeVR.h`

QTVRGetConstraints

Obtains the current constraints of a QuickTime VR movie.

```
OSErr QTVRGetConstraints (
   QTVRInstance qtvr,
   UInt16 kind,
   float *minValue,
   float *maxValue
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

kind

The type of constraints to be returned (see below). See these constants:

`kQTVRPan`

`kQTVRTilt`

`kQTVRFieldOfView`

minValue

On entry, a pointer to a floating-point value. On return, the current minimum constraint of the specified type is copied into that value.

maxValue

On entry, a pointer to a floating-point value. On return, the current maximum constraint of the specified type is copied into that value.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function returns, in the floating-point values pointed to by the `minValue` and `maxValue` parameters, the current minimum and maximum constraints of the type specified by the `kind` parameter. The values returned by this function are unaffected by the current control settings.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

vrscript

vrscript.win

Declared In

`QuickTimeVR.h`

QTVRGetConstraintStatus

Obtains the set of constraints active for the current view.

```
UInt32 QTVRGetConstraintStatus (
   QTVRInstance qtvr
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

Return Value

A long integer (see below) whose bits encode the constraints currently active for the QuickTime VR movie specified by the `qtvr` parameter.

Discussion

The values returned by `QTVRGetConstraintStatus` are unaffected by the current control settings.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeVR.h`

QTVRGetControlSetting

Obtains the current state of a control setting for an object node.

```
OSErr QTVRGetControlSetting (
   QTVRInstance qtvr,
   QTVRControlSetting setting,
   Boolean *enable
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

setting

A control setting (see below). See these constants:

`kQTVRWrapPan`

`kQTVRWrapTilt`

`kQTVRCanZoom`

`kQTVRReverseHControl`

`kQTVRReverseVControl`

`kQTVRSwapHVControl`

`kQTVRTranslation`

enable

On entry, a pointer to a Boolean value. On return, that value is set to TRUE if the specified control setting is currently enabled for the specified object node or to FALSE otherwise.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function returns, through the `enable` parameter, the current state of the control setting specified by the `setting` parameter for the object node specified by the `qtvr` parameter. If enable is TRUE, the specified setting is currently enabled; otherwise, the setting is disabled.

Special Considerations
`QTVRGetControlSetting` is valid only for object nodes.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
vrbackbuffer.win

vrcursors

vrmakeobject

vrmovies

vrscript.win

Declared In
`QuickTimeVR.h`

QTVRGetCurrentMouseMode

Obtains the current mouse control modes.

```
UInt32 QTVRGetCurrentMouseMode (
   QTVRInstance qtvr
);
```

Parameters

`qtvr`
: An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

Return Value
A constant (see below) that describes the current mouse control modes.

Discussion
The value returned by this function is an unsigned long integer that encodes the current mouse control modes. If a bit in the integer is set, the corresponding mode is one of the current mouse modes. The mode bits are addressed using the above constants. Notice that several modes can be returned. That means a return value could have both zooming and translating set, for example.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeVR.h`

QTVRGetCurrentNodeID

Obtains the current node of a movie.

```
UInt32 QTVRGetCurrentNodeID (
   QTVRInstance qtvr
);
```

Parameters

qtvr
An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

Return Value
The ID of the current node of the specified movie.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
vrbackbuffer.win
vrcursors
vrmovies
vrscript
vrscript.win

Declared In
`QuickTimeVR.h`

QTVRGetCurrentViewDuration

Obtains the duration of the current view of an object node.

```
TimeValue QTVRGetCurrentViewDuration (
   QTVRInstance qtvr
);
```

Parameters

qtvr
An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

Return Value
The duration of the current view of the object node specified by the `qtvr` parameter.

Special Considerations
This function is valid only for object nodes. You cannot change a node's view duration.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeVR.h`

QTVRGetFieldOfView

Obtains the vertical field of view of a QuickTime VR movie.

```
float QTVRGetFieldOfView (
   QTVRInstance qtvr
);
```

Parameters
qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

Return Value
The current vertical field of view of the QuickTime VR movie specified by the `qtvr` parameter. The vertical field of view is a floating-point value that specifies the angle created by the two lines that connect the viewpoint to the top and bottom of the image.

Discussion
The following code fragment illustrates the use of this function:

```
// QTVRGetFieldOfView coding example
#define kDirIn       4L
#define kDirOut      5L
void MyZoomInOrOut (QTVRInstance theInstance, long theDir)
{
    float    theFloat;
    theFloat =QTVRGetFieldOfView(theInstance);
    switch (theDir) {
        case kDirIn:
            theFloat =theFloat / 2.0;
            break;
        case kDirOut:
            theFloat =theFloat * 2.0;
            break;
        default:
            break;
    }
    QTVRSetFieldOfView(theInstance, theFloat);
    QTVRUpdate(theInstance, kQTVRStatic);
}
```

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
vrscript

vrscript.win

vrspeech

Declared In
`QuickTimeVR.h`

QTVRGetFrameAnimation

Obtains the current state of frame animation for an object node.

```
Boolean QTVRGetFrameAnimation (
   QTVRInstance qtvr
);
```

Parameters
`qtvr`

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

Return Value
TRUE if frame animation is currently enabled, FALSE otherwise.

Special Considerations
This function is valid only for object nodes.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeVR.h`

QTVRGetFrameRate

Obtains the current frame rate of an object node.

```
float QTVRGetFrameRate (
   QTVRInstance qtvr
);
```

Parameters
`qtvr`

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

Return Value
The current frame rate of the object node specified by the `qtvr` parameter. A frame rate is a floating-point value in the range from -100.0 to +100.0.

Discussion
An object node's default frame rate is stored in the movie file.

Special Considerations

`QTVRGetFrameRate` is valid only for object nodes.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
vrscript

vrscript.win

Declared In
`QuickTimeVR.h`

QTVRGetHotSpotRegion

Obtains the region occupied by a hot spot.

```
OSErr QTVRGetHotSpotRegion (
   QTVRInstance qtvr,
   UInt32 hotSpotID,
   RgnHandle hotSpotRegion
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

hotSpotID

A hot spot ID.

hotSpotRegion

On entry, an initialized handle to a region. On return, this region is rewritten with the region occupied by the hot spot having the specified ID.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
The returned region is clipped to the bounds of the movie's graphics world. You can obtain the regions of all visible hot spots by calling `QTVRGetVisibleHotSpots` (page 2018) and then calling this function for each hot spot ID in the list.

Special Considerations

The first time you call this function, a significant amount of memory might need to be allocated. Accordingly, you should always check for Memory Manager errors returned by this function. Your application is responsible for disposing of the memory occupied by the returned region.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
vrscript

vrscript.win

Declared In
`QuickTimeVR.h`

QTVRGetHotSpotType

Obtains the type of a QuickTime VR hot spot.

```
OSErr QTVRGetHotSpotType (
   QTVRInstance qtvr,
   UInt32 hotSpotID,
   OSType *hotSpotType
);
```

Parameters

`qtvr`
: An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

`hotSpotID`
: A hot spot ID.

`hotSpotType`
: On entry, a pointer to a long integer. On return, that long integer contains the type of the hot spot specified by the hot spot ID.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function gets the type of a hot spot whose ID you specify. In combination with the `kQTVRGetHotSpotTypeSelector` intercept selector (see `QTVRInstallInterceptProc` (page 2021)), this allows an application to change a hot spot's type dynamically. For example, an application can take an existing movie and cause VR to display the cursors for a type of hotspot different from the one the movie was originally authored with. In combination with intercepting `kQTVRTriggerHotSpotSelector`, this would allow an Internet plugin to override undefined or link hotspots in movies to make them appear and act as though they are URL links instead. If `kQTVRTriggerHotSpotSelector` is not intercepted, VR will attempt to act on the hotspot in the normal way; by storing both link and URL data in a file, the exact behavior can be determined at runtime by an application to allow linking to either another node locally or a remote URL link, depending on system configuration or other arbitrary considerations.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code

vrcursors

vrcursors.win

vrscript

vrscript.win

Declared In

`QuickTimeVR.h`

QTVRGetImagingProperty

Obtains the current value of an imaging property of a movie.

```
OSErr QTVRGetImagingProperty (
   QTVRInstance qtvr,
   QTVRImagingMode imagingMode,
   UInt32 imagingProperty,
   SInt32 *propertyValue
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

imagingMode

An imaging mode (see below). See these constants:

`kQTVRStatic`

`kQTVRMotion`

`kQTVRAllModes`

imagingProperty

An imaging property (see below). See these constants:

`kQTVRImagingCorrection`

`kQTVRImagingQuality`

`kQTVRImagingDirectDraw`

`kQTVRImagingCurrentMode`

propertyValue

On entry, a pointer to a long integer. On return, that long integer contains the current value of the specified imaging property for the specified mode.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function returns, in the long integer pointed to by the `propertyValue` parameter, the current value of the property specified by the `imagingProperty` parameter when the QuickTime VR movie specified by the `qtvr` parameter is in the mode specified by the `imagingMode` parameter.

Special Considerations

`QTVRGetImagingProperty` is valid only for panoramic nodes.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeVR.h`

QTVRGetInteractionProperty

Obtains the value of an interaction property.

```
OSErr QTVRGetInteractionProperty (
   QTVRInstance qtvr,
   UInt32 property,
   void *value
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

property

An interaction property type (see below). See these constants:

`kQTVRInteractionMouseClickHysteresis`

`kQTVRInteractionMouseClickTimeout`

`kQTVRInteractionPanTiltSpeed`

`kQTVRInteractionZoomSpeed`

`kQTVRInteractionTranslateOnMouseDown`

`kQTVRInteractionMouseMotionScale`

`kQTVRInteractionNudgeMode`

value

On entry, a pointer to a block of memory. On return, that memory contains the current value of the specified interaction property.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function returns, in the block of memory pointed to by the `value` parameter, the current value of the property specified by the `property` parameter for the QuickTime VR movie specified by the `qtvr` parameter. That block of memory must be large enough to hold the returned value.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeVR.h`

QTVRGetMouseDownTracking

Obtains the current state of mouse-down tracking.

```
Boolean QTVRGetMouseDownTracking (
   QTVRInstance qtvr
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

Return Value

A Boolean value that indicates whether QuickTime VR is currently handling mouse-down tracking for the QuickTime VR movie specified by the `qtvr` parameter (TRUE) or not (FALSE).

Discussion

By default, QuickTime VR tracks mouse clicks in a QuickTime VR movie and triggers hot spots as necessary.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeVR.h`

QTVRGetMouseOverTracking

Obtains the current state of mouse-over tracking.

```
Boolean QTVRGetMouseOverTracking (
   QTVRInstance qtvr
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

Return Value

A Boolean value that indicates whether QuickTime VR is currently handling mouse-over tracking for the QuickTime VR movie specified by the `qtvr` parameter (TRUE) or not (FALSE).

Discussion

By default, QuickTime VR tracks mouse movements in a QuickTime VR movie and changes the shape of the cursor as appropriate.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeVR.h`

QTVRGetNodeInfo

Obtains the node information atom container that describes a node and all the hot spots in the node.

```
OSErr QTVRGetNodeInfo (
   QTVRInstance qtvr,
   UInt32 nodeID,
   QTAtomContainer *nodeInfo
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

nodeID

A node ID. Set this parameter to `kQTVRCurrentNode` to receive information about the current node.

nodeInfo

On return, a pointer to an atom container that contains information about the specified node.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function returns, in the `nodeInfo` parameter, a pointer to an atom container that contains information about the node specified by the `nodeID` parameter in the movie specified by the `qtvr` parameter. The atom container includes information about all the hot spots contained in that node. You can use the QuickTime atom functions to extract atoms from that container. You can also use those functions to access the hot spot atom list. All hot spot atoms are contained in the hot spot parent atom.

Special Considerations

The node information atom container returned by this function is a copy of the atom container maintained internally by the QuickTime VR Manager. You should dispose of the node information atom container, by calling `QTDisposeAtomContainer` (page 1427), when you're finished using it.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

vrbackbuffer.win

vrcursors

vrmakeobject

vrmovies

vrscript.win

Declared In

`QuickTimeVR.h`

QTVRGetNodeType

Obtains the type of a movie node.

```
OSType QTVRGetNodeType (
   QTVRInstance qtvr,
   UInt32 nodeID
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

nodeID

A node ID. Pass `kQTVRCurrentNode` for the current node.

Return Value

The type of the node specified by the `nodeID` parameter in the QuickTime VR movie specified by the `qtvr` parameter.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

vrbackbuffer

vrbackbuffer.win

vrcursors

vrscript

vrscript.win

Declared In

`QuickTimeVR.h`

QTVRGetPanAngle

Obtains the pan angle of a QuickTime VR movie.

```
float QTVRGetPanAngle (
   QTVRInstance qtvr
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

Return Value

A floating-point value that represents the current pan angle of the QuickTime VR movie specified by the `qtvr` parameter.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
vrbackbuffer.win

vrmovies

vrscript

vrscript.win

vrspeech

Declared In
`QuickTimeVR.h`

QTVRGetQTVRInstance

Obtains an instance of a QuickTime VR movie.

```
OSErr QTVRGetQTVRInstance (
   QTVRInstance *qtvr,
   Track qtvrTrack,
   MovieController mc
);
```

Parameters

qtvr

On return, an instance of the specified QuickTime VR movie.

qtvrTrack

A QTVR track contained in a QuickTime movie. You can obtain a reference to this track by calling `QTVRGetQTVRTrack` (page 2010).

mc

An identifier for the movie controller to be associated with the new QuickTime VR movie instance. You obtain this identifier from `OpenComponent` or `OpenDefaultComponent`, or from `NewMovieController` (page 1392).

Return Value
If `qtvrTrack` does not specify a QTVR track, this function returns `NIL` in the `qtvr` parameter and an error code as its function result. See `Error Codes`. Returns `noErr` if there is no error.

Discussion
You need a QuickTime VR movie instance to call most other QuickTime VR functions. This function returns, in the `qtvr` parameter, an instance of the QuickTime VR movie specified by the `qtvrTrack` parameter. Here's an example of code that gets a QTVR instance:

```
// QTVRGetQTVRInstance coding example
// See "Discovering QuickTime," page 390
QTVRInstance MyGetQTVRInstanceFromMC (MovieController mc)
{
    Track               track =NIL;
    QTVRInstance        qtvrinstance =NIL;
    Movie               movie =NIL;
    //Get the movie from the movie controller.
    movie =MCGetMovie(mc);
    if (movie !=NIL) {
        //Get the first QTVR track in the movie.
        track =QTVRGetQTVRTrack(movie, 1);
        //Get a QTVR instance for that QTVR track.
        if (track !=NIL) {
            QTVRGetQTVRInstance(qtvrinstance, track, mc);
            //Set our units to be degrees.
            if (qtvrinstance !=NIL)
                QTVRSetAngularUnits(qtvrinstance, kQTVRDegrees);
        }
    }
    return qtvrinstance;
}
```

Special Considerations

It's not necessary to dispose of a QuickTime VR movie instance.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTCarbonShell

vrbackbuffer.win

vrcursors

vrscript

vrscript.win

Declared In
`QuickTimeVR.h`

QTVRGetQTVRTrack

Obtains a QTVR track contained in a QuickTime movie to use in the QTVRGetQTVRInstance call.

```
Track QTVRGetQTVRTrack (
   Movie theMovie,
   SInt32 index
);
```

Parameters

`theMovie`
: A QuickTime movie.

`index`
: The index of the desired QTVR track.

Return Value
A track identifier for the QTVR track that has the index specified by the `index` parameter in the QuickTime movie specified by the `theMovie` parameter. If there is no such track, or the movie is not a QTVR movie, this function returns the value `NIL`.

Discussion
Here's an example of using this function to help get an instance of a QTVR movie running in a movie controller:

```
// QTVRGetQTVRTrack coding example
// See "Discovering QuickTime," page 390
QTVRInstance MyGetQTVRInstanceFromMC (MovieController mc)
{
    Track               track =NIL;
    QTVRInstance        qtvrinstance =NIL;
    Movie               movie =NIL;
    //Get the movie from the movie controller.
    movie =MCGetMovie(mc);
    if (movie !=NIL) {
        //Get the first QTVR track in the movie.
        track =QTVRGetQTVRTrack(movie, 1);
        //Get a QTVR instance for that QTVR track.
        if (track !=NIL) {
            QTVRGetQTVRInstance(qtvrinstance, track, mc);
            //Set our units to be degrees.
            if (qtvrinstance !=NIL)
                QTVRSetAngularUnits(qtvrinstance, kQTVRDegrees);
        }
    }
    return qtvrinstance;
}
```

Version Notes
Introduced in QuickTime 3 or earlier. QuickTime VR 2.1 supports files with at most one QTVR track; hence the value for the `index` parameter of a movie made with QuickTime VR 2.1 is always 1. Panorama and object movies made with QuickTime VR version 1.0 have no QTVR track. This function returns the track ID of the panorama track for version 1.0 panorama movies and the track ID of the image video track for version 1.0 object movies.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTCarbonShell

vrbackbuffer.win

vrcursors

vrscript

vrscript.win

Declared In
`QuickTimeVR.h`

QTVRGetTiltAngle

Obtains the tilt angle of a QuickTime VR movie.

```
float QTVRGetTiltAngle (
   QTVRInstance qtvr
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

Return Value

A floating-point value that represents the current tilt angle of the QuickTime VR movie specified by the `qtvr` parameter.

Discussion

When a cylindrical panorama is zoomed all the way out (to its maximum vertical field of view), it can no longer be tilted because its entire vertical surface is exposed. Attempting to set the tilt angle will result in a `ConstraintReachedErr` error (except for the degenerate case of setting the tilt angle to its current value).

The tilt angle may not be zero when the panorama is fully zoomed out; it may be tilted by one line of pixels. The tilt angle is small in this case, typically 0.006, but its exact magnitude depends on the height of the panorama; the taller the panorama, the smaller the error.

Do not test for a tilt angle of exactly 0.0 or attempt to adjust the tilt angle of a fully zoomed-out panorama.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

vrbackbuffer.win

vrmovies

vrscript

vrscript.win

vrspeech

Declared In

`QuickTimeVR.h`

QTVRGetViewAnimation

Obtains the current state of view animation for an object node.

```
Boolean QTVRGetViewAnimation (
   QTVRInstance qtvr
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

Return Value

A Boolean value that indicates the current state of view animation for the object node specified by the `qtvr` parameter. It is TRUE if view animation is enabled, FALSE otherwise.

Special Considerations

This function is valid only for object nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeVR.h`

QTVRGetViewCenter

Obtains the view center of a QuickTime VR movie.

```
OSErr QTVRGetViewCenter (
   QTVRInstance qtvr,
   QTVRFloatPoint *viewCenter
);
```

Parameters

`qtvr`

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

`viewCenter`

On entry, a pointer to a `QTVRFloatPoint` structure. On return, that structure contains the current view center of the specified movie.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Special Considerations

This function is valid only for object nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeVR.h`

QTVRGetViewCurrentTime

Obtains the current time in the current view.

```
TimeValue QTVRGetViewCurrentTime (
   QTVRInstance qtvr
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

Return Value

The current time in the current view of the object node specified by the `qtvr` parameter. The returned value is always greater than or equal to 0 and less than or equal to the value returned by `QTVRGetCurrentViewDuration` (page 1999).

Special Considerations

`QTVRGetViewCurrentTime` is valid only for object nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

vrscript

vrscript.win

Declared In

`QuickTimeVR.h`

QTVRGetViewingLimits

Obtains the current viewing limits of a QuickTime VR movie.

```
OSErr QTVRGetViewingLimits (
   QTVRInstance qtvr,
   UInt16 kind,
   float *minValue,
   float *maxValue
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

kind

The type of viewing limits to be returned (see below). See these constants:

`kQTVRPan`

`kQTVRTilt`

`kQTVRFieldOfView`

minValue
: On entry, a pointer to a floating-point value. On return, the minimum viewing limit of the specified type is copied into that value.

maxValue
: On entry, a pointer to a floating-point value. On return, the maximum viewing limit of the specified type is copied into that value.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function returns, in the floating-point values pointed to by the `minValue` and `maxValue` parameters, the current minimum and maximum values for angles whose type is specified by the `kind` parameter. The maximum field of view of a panoramic node can be limited by the size of the back buffer and the current aspect ratio of the movie's graphics world. The values returned by this function are unaffected by the current control settings.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeVR.h`

QTVRGetViewParameter

Undocumented

```
OSErr QTVRGetViewParameter (
   QTVRInstance qtvr,
   UInt32 viewParameter,
   void *value,
   UInt32 flagsIn,
   UInt32 *flagsOut
);
```

Parameters

qtvr
: *Undocumented*

viewParameter
: *Undocumented*

value
: *Undocumented*

flagsIn
: *Undocumented*

flagsOut
: *Undocumented*

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeVR.h`

QTVRGetViewRate

Obtains the current view rate of an object node.

```
float QTVRGetViewRate (
   QTVRInstance qtvr
);
```

Parameters

qtvr
: An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

Return Value
The current view rate of the object node specified by the `qtvr` parameter. A view rate is a floating-point value in the range from -100.0 to +100.0. An object node's default view rate is stored in the movie file.

Special Considerations
This function is valid only for object nodes.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
vrscript

vrscript.win

Declared In
`QuickTimeVR.h`

QTVRGetViewState

Obtains the value of a view state.

```
OSErr QTVRGetViewState (
   QTVRInstance qtvr,
   QTVRViewStateType viewStateType,
   UInt16 *state
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

viewStateType

A view state type (see below). See these constants:

`kQTVRDefault`

`kQTVRCurrent`

`kQTVRMouseDown`

state

On entry, a pointer to a short integer. On return, that integer is set to the current value of the specified type of view state.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Special Considerations

This function is valid only for object nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeVR.h`

QTVRGetViewStateCount

Obtains the number of view states of an object node.

```
UInt16 QTVRGetViewStateCount (
   QTVRInstance qtvr
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

Return Value

The number of view states associated with the object node specified by the `qtvr` parameter. The number of view states in an object movie is defined by the movie file.

Special Considerations

This function is valid only for object nodes.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeVR.h`

QTVRGetVisible

Obtains a movie's visibility state.

```
Boolean QTVRGetVisible (
   QTVRInstance qtvr
);
```

Parameters

qtvr
: An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

Return Value
A Boolean value that indicates whether the QuickTime VR movie specified by the `qtvr` parameter is visible (TRUE) or not (FALSE).

Special Considerations

This function is valid only for panoramic nodes.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeVR.h`

QTVRGetVisibleHotSpots

Obtains a list of the currently visible hot spots in a QuickTime VR movie.

```
UInt32 QTVRGetVisibleHotSpots (
   QTVRInstance qtvr,
   Handle hotSpots
);
```

Parameters

qtvr
: An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

hotSpots

On entry, a valid handle to a block of memory. On return, that block of memory is filled with a list of the IDs of the visible hot spots in the specified QuickTime VR movie. If necessary, the handle is resized to hold all the hot spot IDs. Accordingly, the handle must be unlocked at the time you call this function.

Return Value

The number of hot spot IDs returned though the hotSpots parameter.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

vrscript

vrscript.win

Declared In

`QuickTimeVR.h`

QTVRGetVRWorld

Obtains the VR world atom container for a movie.

```
OSErr QTVRGetVRWorld (
   QTVRInstance qtvr,
   QTAtomContainer *VRWorld
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

VRWorld

On return, a pointer to an atom container that contains information about the specified movie.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function returns, in the `VRWorld` parameter, a pointer to an atom container that contains general scene information about the QuickTime VR movie specified by the `qtvr` parameter, as well as a list of all the nodes in that movie. You can use the QuickTime atom functions to extract atoms from that container.

Special Considerations

The VR world atom container returned by this function is a copy of the atom container maintained internally by the QuickTime VR Manager. You should dispose of the VR world atom container by calling `QTDisposeAtomContainer` (page 1427) when you're finished using it.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code
vrbackbuffer.win
vrcursors
vrmakeobject
vrmovies
vrscript.win

Declared In
`QuickTimeVR.h`

QTVRGoToNodeID

Sets the current node of a movie.

```
OSErr QTVRGoToNodeID (
   QTVRInstance qtvr,
   UInt32 nodeID
);
```

Parameters

qtvr
An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

nodeID
The ID of the node you want to be the current node. The QuickTime VR Manager defines several constants (see below) for specific nodes. See these constants:
`kQTVRCurrentNode`
`kQTVRPreviousNode`
`kQTVRDefaultNode`

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Special Considerations
Setting the current node also sets the pan, tilt, and field of view of the new current node to their default values. As a result, if you wish to set non-default angles, you should call this function before you call `QTVRSetPanAngle` (page 2051), `QTVRSetTiltAngle` (page 2054), or `QTVRSetFieldOfView` (page 2044).

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
vrscript
vrscript.win

Declared In
`QuickTimeVR.h`

QTVRInstallInterceptProc

Installs or removes an intercept procedure for a QuickTime VR Manager function.

```
OSErr QTVRInstallInterceptProc (
   QTVRInstance qtvr,
   QTVRProcSelector selector,
   QTVRInterceptUPP interceptProc,
   SInt32 refCon,
   UInt32 flags
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (page 2009).

selector

A selector (see below) that indicates which QuickTime VR function to intercept. See these constants:

- kQTVRSetPanAngleSelector
- kQTVRSetTiltAngleSelector
- kQTVRSetFieldOfViewSelector
- kQTVRSetViewCenterSelector
- kQTVRMouseEnterSelector
- kQTVRMouseWithinSelector
- kQTVRMouseLeaveSelector
- kQTVRMouseDownSelector
- kQTVRMouseStillDownSelector
- kQTVRMouseUpSelector
- kQTVRTriggerHotSpotSelector
- kQTVRGetHotSpotTypeSelector

interceptProc

A Universal Procedure Pointer for a `QTVRInterceptProc` callback. Set this parameter to `NIL` to remove a previously installed intercept procedure.

refCon

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

flags

Unused. Set this parameter to 0.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function installs the procedure specified by the `interceptProc` parameter as an intercept procedure for the QuickTime VR function specified by the `selector` parameter for the QuickTime VR movie specified by the `qtvr` parameter. Your intercept procedure is called whenever QuickTime VR is about to execute the function you are intercepting.

Your procedure can simply replace the intercepted function, by returning TRUE in the `cancel` parameter; or it can call through to the intercepted function, by calling `QTVRCallInterceptedProc` (page 1984); or it can allow the intercepted function to execute when the intercept procedure returns, by returning FALSE in the `cancel` parameter.

Here's an example of using this function:

```
// QTVRInstallInterceptProc coding example
// See "Discovering QuickTime," page 398
QTVRInterceptUPP MyInstallInterceptProcedure (QTVRInstance qtvrinstance)
{
    QTVRInterceptUPP    lpfnIntercept;
    lpfnIntercept =NewQTVRInterceptProc(MyInterceptProc);
    QTVRInstallInterceptProc(qtvrinstance, kQTVRSetPanAngleSelector,
                                                lpfnIntercept, 0, 0);
    return lpfnIntercept
}
```

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
vrscript

vrscript.win

vrspeech

Declared In
`QuickTimeVR.h`

QTVRInteractionNudge

Translates the image and displays the new view or rotates the object in a particular direction and displays its new appearance.

```
OSErr QTVRInteractionNudge (
   QTVRInstance qtvr,
   QTVRNudgeControl direction
);
```

Parameters

qtvr
: An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

direction

The direction of the nudge (see below). The type of adjustment depends on the nudge interaction mode, which you can set with `QTVRSetInteractionProperty` (page 2047). If the nudge interaction mode is `kQTVRNudgeRotate`, the action of `QTVRInteractionNudge` is to rotate the object in the specified direction. If the nudge interaction mode is `kQTVRNudgeTranslate`, the action of `QTVRInteractionNudge` is to translate the image in the specified direction. If the nudge interaction mode is `kQTVRUNudgeSameAsMouse`, the action of `QTVRInteractionNudge` is determined by the current mouse mode, which you can determine by calling `QTVRGetCurrentMouseMode` (page 1998). See these constants:

`kQTVRRight`

`kQTVRUpRight`

`kQTVRUp`

`kQTVRUpLeft`

`kQTVRLeft`

`kQTVRDown`

`kQTVRDownRight`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function adjusts the current view of the movie specified by the `qtvr` parameter as indicated by the `direction` parameter. The type of adjustment depends on the property setting for nudge interaction mode, which you can set with `QTVRSetInteractionProperty` (page 2047).

Special Considerations

This function is valid only for object nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeVR.h`

QTVRMouseDown

Handles the user's clicking the mouse button when the cursor is in a QuickTime VR movie for which mouse-down tracking is disabled.

```
OSErr QTVRMouseDown (
   QTVRInstance qtvr,
   Point pt,
   UInt32 when,
   UInt16 modifiers,
   UInt32 *hotSpotID,
   WindowRef w
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (page 2009).

pt

The current location of the cursor, in the local coordinates of the graphics world specified by the w parameter.

when

The time, in the number of ticks (sixtieths of a second) since system startup, when the mouse-down event was posted.

modifiers

A short integer, each bit of which is represented by a constant (see below) that provides information about the state of the modifier keys and the mouse button at the time the event was posted. More than one bit may be set. See these constants:

hotSpotID

On entry, a pointer to a long integer. On return, that long integer contains the ID of the hot spot that lies beneath the specified point, or the value 0 if no hot spot lies beneath that point.

w

A pointer to a graphics world.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function returns, in the long integer pointed to by the `hotSpotID` parameter, the ID of the hot spot in the QuickTime VR movie specified by the `qtvr` parameter that lies directly under the point specified by the `pt` parameter. If no hot spot lies under that point, the long integer is set to 0. `QTVRMouseDown` also performs any other tasks that are typically performed when the user clicks the mouse button when the cursor is in a QuickTime VR movie.

Special Considerations

You need to call `QTVRMouseDown` only if you have disabled mouse-down tracking for the specified QuickTime VR movie.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeVR.h`

QTVRMouseEnter

Handles the user's moving the cursor into a QuickTime VR movie for which mouse-over tracking is disabled.

```
OSErr QTVRMouseEnter (
   QTVRInstance qtvr,
   Point pt,
   UInt32 *hotSpotID,
   WindowRef w
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

pt

The current location of the cursor, in the local coordinates of the graphics world specified by the `w` parameter.

hotSpotID

On entry, a pointer to a long integer. On return, that long integer contains the ID of the hot spot that lies beneath the specified point, or the value 0 if no hot spot lies beneath that point.

w

A pointer to a graphics world.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function returns, in the long integer pointed to by the `hotSpotID` parameter, the ID of the hot spot in the QuickTime VR movie specified by the `qtvr` parameter that lies directly under the point specified by the `pt` parameter. If no hot spot lies under that point, the long integer is set to 0. `QTVRMouseEnter` also performs any other tasks that are typically performed when the user first moves the cursor into a QuickTime VR movie.

Special Considerations

You need to call `QTVRMouseEnter` only if you have disabled mouse-over tracking for the specified QuickTime VR movie.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeVR.h`

QTVRMouseLeave

Handles the user's moving the cursor out of a QuickTime VR movie for which mouse-over tracking is disabled.

```
OSErr QTVRMouseLeave (
   QTVRInstance qtvr,
   Point pt,
   WindowRef w
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

pt

The current location of the cursor, in the local coordinates of the graphics world specified by the `w` parameter.

w

A pointer to a graphics world.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function performs any tasks that are typically performed when the user moves the cursor out of a QuickTime VR movie.

Special Considerations

You need to call `QTVRMouseLeave` only if you have disabled mouse-over tracking for the specified QuickTime VR movie.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeVR.h`

QTVRMouseStillDown

Handles the user's holding down the mouse button while the cursor is in a QuickTime VR movie for which mouse-down tracking is disabled.

```
OSErr QTVRMouseStillDown (
   QTVRInstance qtvr,
   Point pt,
   UInt32 *hotSpotID,
   WindowRef w
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

pt

The current location of the cursor, in the local coordinates of the graphics world specified by the `w` parameter.

hotSpotID

On entry, a pointer to a long integer. On return, that long integer contains the ID of the hot spot that lies beneath the specified point, or the value 0 if no hot spot lies beneath that point.

w

A pointer to a graphics world.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function returns, in the long integer pointed to by the `hotSpotID` parameter, the ID of the hot spot in the QuickTime VR movie specified by the `qtvr` parameter that lies directly under the point specified by the `pt` parameter. If no hot spot lies under that point, the long integer is set to 0. This function also performs any other tasks that are typically performed when the user holds down the mouse button when the cursor is in a QuickTime VR movie. You should call this function repeatedly for as long as the user holds down the mouse button while the cursor is in the specified QuickTime VR movie.

Special Considerations

You need to call this function only if you have disabled mouse-down tracking for the specified QuickTime VR movie. Applications running on operating systems other than Mac OS should use the extended form of this function, `QTVRMouseStillDownExtended` (page 2027).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeVR.h`

QTVRMouseStillDownExtended

Handles the user's holding down the mouse button while the cursor is in a QuickTime VR movie for which mouse-down tracking is disabled.

```
OSErr QTVRMouseStillDownExtended (
   QTVRInstance qtvr,
   Point pt,
   UInt32 *hotSpotID,
   WindowRef w,
   UInt32 when,
   UInt16 modifiers
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

pt
: The current location of the cursor, in the local coordinates of the graphics world specified by the `w` parameter.

hotSpotID
: On entry, a pointer to a long integer. On return, that long integer contains the ID of the hot spot that lies beneath the specified point, or the value 0 if no hot spot lies beneath that point.

w
: A pointer to a graphics world.

when
: The current time as the number of ticks (sixtieths of a second) since system startup.

modifiers
: A short integer, each bit of which is represented by a constant (see below) that provides information about the state of the modifier keys and the mouse button at the time the event was posted. More than one bit may be set. See these constants:

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function uses the same intercept as `QTVRMouseStillDown` (page 2026) but has two additional parameters. Applications that intercept this function should always check the `paramCount` field to make sure it is 5 before using the last two fields. You should call this function repeatedly for as long as the user holds down the mouse button while the cursor is in the specified QuickTime VR movie.

Special Considerations

You need to call this function only if you have disabled mouse-down tracking for the specified QuickTime VR movie. Internally, QuickTime VR always uses this function instead of `QTVRMouseStillDown`. Developers implementing their own mouse down tracking don't need to use the extended version unless they also intercept the procedure and need the added parameters.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeVR.h`

QTVRMouseUp

Handles the user's releasing the mouse button while the cursor is in a QuickTime VR movie for which mouse-down tracking is disabled.

```
OSErr QTVRMouseUp (
   QTVRInstance qtvr,
   Point pt,
   UInt32 *hotSpotID,
   WindowRef w
);
```

Parameters

qtvr
: An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

pt
: The current location of the cursor, in the local coordinates of the graphics world specified by the `w` parameter.

hotSpotID
: On entry, a pointer to a long integer. On return, that long integer contains the ID of the hot spot that lies beneath the specified point, or the value 0 if no hot spot lies beneath that point.

w
: A pointer to a graphics world.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function returns, in the long integer pointed to by the `hotSpotID` parameter, the ID of the hot spot in the QuickTime VR movie specified by the `qtvr` parameter that lies directly under the point specified by the `pt` parameter. If no hot spot lies under that point, the long integer is set to 0. this function also performs any other tasks that are typically performed when the user releases the mouse button after clicking it when the cursor is in a QuickTime VR movie.

Special Considerations

You need to call this function only if you have disabled mouse-down tracking for the specified QuickTime VR movie. Applications running on operating systems other than Mac OS should use the extended form of this function, `QTVRMouseUpExtended` (page 2029).

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeVR.h`

QTVRMouseUpExtended

Handles the user's releasing the mouse button while the cursor is in a QuickTime VR movie for which mouse-down tracking is disabled.

```
OSErr QTVRMouseUpExtended (
   QTVRInstance qtvr,
   Point pt,
   UInt32 *hotSpotID,
   WindowRef w,
   UInt32 when,
   UInt16 modifiers
);
```

Parameters

qtvr
: An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

pt
: The current location of the cursor, in the local coordinates of the graphics world specified by the `w` parameter.

hotSpotID
: On entry, a pointer to a long integer. On return, that long integer contains the ID of the hot spot that lies beneath the specified point, or the value 0 if no hot spot lies beneath that point.

w
: A pointer to a graphics world.

when
: The time, in the number of ticks (sixtieths of a second) since system startup, when the mouse-up event was posted.

modifiers
: A short integer, each bit of which is represented by a constant (see below) that provides information about the state of the modifier keys and the mouse button at the time the event was posted. More than one bit may be set. See these constants:

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function uses the same intercept as the `QTVRMouseUp` function but has two additional parameters. Applications that intercept this function should always check the `paramCount` field to make sure it is 5 before using the last two fields.

Special Considerations

You need to call `QTVRMouseUp` (page 2028) or this function only if you have disabled mouse-down tracking for the specified QuickTime VR movie. Internally, QuickTime VR always uses the this function function instead of `QTVRMouseUp`. Developers implementing their own mouse down tracking don't need to use the extended version unless they also intercept the procedure and need the added parameters.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeVR.h`

QTVRMouseWithin

Handles the user's leaving the cursor in a QuickTime VR movie for which mouse-over tracking is disabled.

```
OSErr QTVRMouseWithin (
   QTVRInstance qtvr,
   Point pt,
   UInt32 *hotSpotID,
   WindowRef w
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

pt

The current location of the cursor, in the local coordinates of the graphics world specified by the `w` parameter.

hotSpotID

On entry, a pointer to a long integer. On return, that long integer contains the ID of the hot spot that lies beneath the specified point, or the value 0 if no hot spot lies beneath that point.

w

A pointer to a graphics world.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

You should call this function repeatedly for as long as the cursor remains in the specified QuickTime VR movie.

Special Considerations

You need to call `QTVRMouseWithin` only if you have disabled mouse-over tracking for the specified QuickTime VR movie.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeVR.h`

QTVRNudge

Turns one step in a particular direction and displays the new view.

```
OSErr QTVRNudge (
   QTVRInstance qtvr,
   QTVRNudgeControl direction
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

direction

The direction of the nudge (see below). Any value of the `direction` parameter that is not predefined is mapped to the closest defined value. See these constants:

`kQTVRRight`

`kQTVRUpRight`

`kQTVRUp`

`kQTVRUpLeft`

`kQTVRLeft`

`kQTVRDown`

`kQTVRDownRight`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function adjusts the current view of the movie specified by the `qtvr` parameter as indicated by the `direction` parameter. In particular, it turns one step in the indicated direction and displays the new view. For example, to move to the next view that is right and up from the current view, set the `direction` parameter to `kQTVRUpRight` (that is, pi/4 radians, or 45 degrees). For objects, if no view is located at the adjacent object view defined by the nudge direction and wrapping is off in the desired direction, then this function remains at the current view and returns the result code `constraintReachedErr`. For objects, this function is useful for changing to an adjacent view without having to know the new pan and tilt angles. The direction of the nudge is affected by the current control settings.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

vrscript

vrscript.win

Declared In

`QuickTimeVR.h`

QTVRPanToColumn

Obtains the column number in the object image array that corresponds to a pan angle.

```
short QTVRPanToColumn (
   QTVRInstance qtvr,
   float panAngle
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

panAngle

A pan angle.

Return Value

The zero-based column number in the current object image array that corresponds to the pan angle specified by the `panAngle` parameter

Special Considerations

This function is valid only for object nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeVR.h`

QTVRPtToAngles

Obtains the pan and tilt angles of a point.

```
OSErr QTVRPtToAngles (
   QTVRInstance qtvr,
   Point pt,
   float *panAngle,
   float *tiltAngle
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

pt

A point, in the local coordinates of the graphics world of the specified movie.

panAngle

On entry, a pointer to a floating-point value. On return, that value contains the pan angle of the specified point.

tiltAngle

On entry, a pointer to a floating-point value. On return, that value contains the tilt angle of the specified point.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
For a panorama, each point in the current view corresponds to a particular pan and tilt angle, with the point at the center of the view corresponding to the panorama's current pan and tilt angle. This function returns, in the floating-point values pointed to by the `panAngle` and `tiltAngle` parameters, the pan and tilt angles of the point specified by the `pt` parameter.

Special Considerations

This function is valid only for panoramic nodes.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeVR.h`

QTVRPtToHotSpotID

Obtains the ID of the hot spot, if any, that lies beneath a point.

```
OSErr QTVRPtToHotSpotID (
   QTVRInstance qtvr,
   Point pt,
   UInt32 *hotSpotID
);
```

Parameters

`qtvr`

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

`pt`

A point, in the local coordinates of the graphics world of the specified movie.

`hotSpotID`

On entry, a pointer to a long integer. On return, that long integer contains the ID of the hot spot that lies beneath the specified point, or the value 0 if no hot spot lies beneath that point.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeVR.h`

QTVRRefreshBackBuffer

Refreshes the QTVR back buffer.

```
OSErr QTVRRefreshBackBuffer (
   QTVRInstance qtvr,
   UInt32 flags
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

flags

Unused. Set this parameter to 0.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function refreshes some or all of the back buffer associated with the QuickTime VR movie specified by the `qtvr` parameter by reloading the appropriate data from the diced frames in the panorama image track. You can call this function either in a back buffer imaging procedure or elsewhere in your application. If you call this function in a back buffer imaging procedure, only the current rectangle (that is, the rectangle specified by the procedure's `drawRect` parameter) is refreshed. If you call this function outside of a back buffer imaging procedure, all areas of interest specified in the most recent call to `QTVRSetBackBufferImagingProc` (page 2039) are refreshed.

Special Considerations

This function is valid only for panoramic nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

vrbackbuffer.win

vrmovies

vrmovies.win

vrscript

vrscript.win

Declared In

`QuickTimeVR.h`

QTVRReplaceCursor

Replaces any of the standard QuickTime VR cursors with your own custom cursor.

```
OSErr QTVRReplaceCursor (
   QTVRInstance qtvr,
   QTVRCursorRecord *cursRecord
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling QTVRGetQTVRInstance (page 2009).

cursRecord

A pointer to a QTVRCursorRecord structure.

Return Value

See Error Codes. Returns noErr if there is no error.

Discussion

This function replaces one or more of the standard QuickTime VR cursors associated with the instance specified by the qtvr parameter with the cursors specified in the cursor record pointed to by the cursRecord parameter. If the type field of the specified cursor record is kQTVRUseDefaultCursor, the default cursor for the given resource ID is reloaded; in this case, the handle field of that record should be set to NIL.

Special Considerations

This function replaces the standard cursors only for the specified QuickTime VR movie instance. To replace the standard cursors for all QuickTime VR movie instances you create, you need to call this function for each such instance.

Version Notes

Introduced in QuickTime 3 or earlier. Note that QuickTime VR 2.1 makes a copy of the cursor handle specified in the cursor record. The application is responsible for disposing of its own cursor handle.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

vrcursors

vrcursors.win

vrscript

vrscript.win

Declared In

QuickTimeVR.h

QTVRRowToTilt

Obtains the tilt angle that corresponds to a row number in a QTVR object image array.

```
float QTVRRowToTilt (
   QTVRInstance qtvr,
   short row
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

row

A row number.

Return Value

The tilt angle that corresponds to the zero-based row number in the object image array specified by the `row` parameter.

Special Considerations

This function is valid only for object nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeVR.h`

QTVRSetAngularUnits

Sets the type of unit used when specifying QTVR angles.

```
OSErr QTVRSetAngularUnits (
   QTVRInstance qtvr,
   QTVRAngularUnits units
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

units

A constant (see below) that indicates the type of angular units to use. See these constants:

`kQTVRDegrees`

`kQTVRRadians`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function sets the type of angular units to be used in all subsequent QuickTime VR Manager calls for the QuickTime VR movie specified by the `qtvr` parameter to the unit type specified by the `units` parameter.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
vrbackbuffer.win

vrcursors

vrmakeobject

vrmovies

vrspeech

Declared In
`QuickTimeVR.h`

QTVRSetAnimationSetting

Sets the state of an animation setting for an object node.

```
OSErr QTVRSetAnimationSetting (
   QTVRInstance qtvr,
   QTVRObjectAnimationSetting setting,
   Boolean enable
);
```

Parameters

`qtvr`
: An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

`setting`
: An animation setting (see below). See these constants:
  - `kQTVRPalindromeViewFrames`
  - `kQTVRDontLoopViewFrames`
  - `kQTVRPlayEveryViewFrame`
  - `kQTVRSyncViewToFrameRate`
  - `kQTVRPalindromeViews`
  - `kQTVRPlayStreamingViews`

`enable`
: A Boolean value that indicates whether the specified animation setting is to be enabled for the specified object node (TRUE) or disabled (FALSE).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Special Considerations

This function is valid only for object nodes.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
vrscript

vrscript.win

Declared In
`QuickTimeVR.h`

QTVRSetBackBufferImagingProc

Installs or removes a QTVR back buffer imaging procedure.

```
OSErr QTVRSetBackBufferImagingProc (
   QTVRInstance qtvr,
   QTVRBackBufferImagingUPP backBufferImagingProc,
   UInt16 numAreas,
   QTVRAreaOfInterest areasOfInterest[],
   SInt32 refCon
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

backBufferImagingProc

A Universal Procedure Pointer for a `QTVRBackBufferImagingProc` callback. To remove a previously installed back buffer imaging procedure, pass `NIL`.

numAreas

The number of area of interest structures in the array pointed to by the `areasOfInterest` parameter.

areasOfInterest[]

A pointer to an array of `QTVRAreaOfInterest` structures. Each structure defines a rectangular areas about which you want your back buffer imaging procedure to be notified. Your procedure is called for each area of interest as it becomes visible or not visible. You indicate when you want your procedure to be called for a particular area of interest by setting flags in the `flags` field in the corresponding area of interest structure. The width of the area of interest is limited by the size of the back buffer. If the back buffer is less than the full cache size, then the area of interest can be no wider than half the size of the back buffer. (For vertical cylinder geometries, limiting factor would be the height of the buffer.) For a full cache back buffer, the width of the area of interest can be the full size of the buffer. If the width limit is exceeded, this function returns `constraintReachedErr`.

refCon

A reference constant. This value is passed to the specified back buffer imaging callback. Use this parameter to point to a data structure containing any information your callback needs.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function installs the procedure specified by the `backBufferImagingProc` parameter as a back buffer imaging procedure for the panoramic node specified by the `qtvr` parameter. You can use that procedure to draw directly into the back buffer. Coordinates in the back buffer are dependent on the current correction

mode; as a result, you need to indicate the area you're interested in drawing into by specifying a pan angle and tilt angle to determine the upper-left corner of the area and a height and width relative to that corner. Specifying a height and width instead of a second pair of pan and tilt angles for the bottom-right coordinate allows the rectangle to wrap around the edge of the panorama.

Special Considerations

This function is valid only for panoramic nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

vrbackbuffer.win

vrmovies

vrmovies.win

vrscript

vrscript.win

Declared In

`QuickTimeVR.h`

QTVRSetBackBufferPrefs

Sets the resolution, pixel format, and size of the back buffer maintained internally by QuickTime VR for caching a panoramic image in a particular pixel format.

```
OSErr QTVRSetBackBufferPrefs (
   QTVRInstance qtvr,
   UInt32 geometry,
   UInt16 resolution,
   UInt32 cachePixelFormat,
   SInt16 cacheSize
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

geometry

The type and orientation of the panorama data (see below). See these constants:

`kQTVRUseMovieGeometry`

`kQTVRVerticalCylinder`

resolution

The desired image resolution (see below). See these constants:

`kQTVRDefaultRes`

`kQTVRFullRes`

`kQTVRHalfRes`

`kQTVRQuarterRes`

cachePixelFormat
: The desired pixel format for the back buffer (see below). See these constants:

cacheSize
: The desired size for the panorama back buffer (see below). See these constants:
  `kQTVRMinimumCache`
  `kQTVRSuggestedCache`
  `kQTVRFullCache`

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function sets the resolution, pixel format, and size of the panorama back buffer for the movie specified by the `qtvr` parameter to the values specified by the `resolution` parameter an the `cachePixelFormat` and `cacheSize` parameters. You can specify a resolution that isn't contained in the movie file; if you do so, QuickTime VR takes the highest resolution image in the file and reduces it to fit into the specified buffer size. If you specify an unsupported pixel format, this function may return an error.

Special Considerations

This function is valid only for panoramic nodes.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
vrscript

vrscript.win

Declared In
`QuickTimeVR.h`

QTVRSetConstraints

Sets the constraints of a VR movie.

```
OSErr QTVRSetConstraints (
   QTVRInstance qtvr,
   UInt16 kind,
   float minValue,
   float maxValue
);
```

Parameters

qtvr
: An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

kind

The type of constraint to set (see below). See these constants:

`kQTVRPan`

`kQTVRTilt`

`kQTVRFieldOfView`

minValue

A floating-point value that contains the desired minimum constraint of the specified type.

maxValue

A floating-point value that contains the desired maximum constraint of the specified type.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function sets the minimum and maximum constraints of the type specified by the `kind` parameter to the values specified by the `minValue` and `maxValue` parameters. Note that when you want to specify a pan angle constraint, the `minValue` and `maxValue` parameters should be specified so that a clockwise sweep from `minValue` to `maxValue` selects the desired angular expanse. For example, to constrain panning in the 90-degree expanse that spreads out 45 degrees on each side of the pan angle 0 degrees, you should set the `minValue` parameter to 315 degrees and the `maxValue` parameter to 45 degrees. Similarly, to constrain panning in the remaining 270-degree expanse, you should set the `minValue` parameter to 45 degrees and the `maxValue` parameter to 315 degrees.

Special Considerations

The values passed to this function are unaffected by the current control settings.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

vrscript

vrscript.win

Declared In

`QuickTimeVR.h`

QTVRSetControlSetting

Sets the state of a control setting for a QTVR object node.

```
OSErr QTVRSetControlSetting (
   QTVRInstance qtvr,
   QTVRControlSetting setting,
   Boolean enable
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

setting

A control setting (see below). See these constants:

`kQTVRWrapPan`

`kQTVRWrapTilt`

`kQTVRCanZoom`

`kQTVRReverseHControl`

`kQTVRReverseVControl`

`kQTVRSwapHVControl`

`kQTVRTranslation`

enable

A Boolean value that indicates whether the specified control setting is to be enabled for the specified object node (TRUE) or disabled (FALSE).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Special Considerations

This function is valid only for object nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

vrscript

vrscript.win

Declared In

`QuickTimeVR.h`

QTVRSetEnteringNodeProc

Installs or removes a node-entering procedure.

```
OSErr QTVRSetEnteringNodeProc (
   QTVRInstance qtvr,
   QTVREnteringNodeUPP enteringNodeProc,
   SInt32 refCon,
   UInt32 flags
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

enteringNodeProc

A Universal Procedure Pointer for a `QTVREnteringNodeProc` callback. To remove a previously installed `QTVREnteringNodeProc` callback, set `enteringNodeProc` to `NIL`.

refCon

A reference constant. This value is passed to the specified node-entering callback. Use this parameter to point to a data structure containing any information your callback needs.

flags

Unused. Set this parameter to 0.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function installs the procedure specified by the `enteringNodeProc` parameter as a node-entering procedure for the QuickTime VR movie specified by the `qtvr` parameter. Your procedure is called whenever a node is entered (either in response to user actions or in response to QuickTime VR Manager functions that change nodes). The reference constant specified by the `refCon` parameter is passed unchanged to that node-entering procedure.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

vrscript

vrscript.win

vrspeech

Declared In

`QuickTimeVR.h`

QTVRSetFieldOfView

Sets the vertical field of view of a QuickTime VR movie.

```
OSErr QTVRSetFieldOfView (
   QTVRInstance qtvr,
   float fieldOfView
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

fieldOfView

The desired vertical field of view for the specified movie. This value is constrained by the maximum field of view of the movie. Values that lie outside that limit are clipped to the maximum. Pan and tilt angle values are also clipped if, when combined with the current field of view, they would cause an image to lie outside the current constraints.

Return Value

See `Error Codes`. Returns `noErr` if there is no error. If the control setting `kQTVRCanZoom` is disabled, the field of view is unchanged and this function returns the result code `constraintReachedErr`. You can use `QTVRSetControlSetting` (page 2042) to control the setting of `kQTVRCanZoom`.

Special Considerations

The pan and tilt angles are subject to the current pan and tilt range constraints, as imposed by the viewing limits and the current field of view. Accordingly, if you want to change the field of view, you should do so before adjusting the pan or tilt angles. Otherwise, the pan and tilt angles are clipped against the current field of view, which may result in an incorrect view when you alter the field of view.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

vrscript

vrscript.win

vrspeech

Declared In

`QuickTimeVR.h`

QTVRSetFrameRate

Sets the frame rate of an object node.

```
OSErr QTVRSetFrameRate (
   QTVRInstance qtvr,
   float rate
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

rate

The desired frame rate of the specified movie. A frame rate is a floating-point value in the range from -100.0 to +100.0. Positive values indicate forward rates, and negative values indicate reverse rates. Set this parameter to 0 to stop the movie. If the value specified lies outside the valid range, this function returns the result code `constraintReachedErr` and sets the frame rate to the nearest constraint.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function is most useful when an object is being viewed with a looping animation. (The current view of the object may contain frames that are played in a loop, as specified by the file format.) You can use this function to change the frame rate of the loop.

Special Considerations

This function is valid only for object nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

vrscript

vrscript.win

Declared In

`QuickTimeVR.h`

QTVRSetImagingProperty

Sets the value of an imaging property of a movie.

```
OSErr QTVRSetImagingProperty (
   QTVRInstance qtvr,
   QTVRImagingMode imagingMode,
   UInt32 imagingProperty,
   SInt32 propertyValue
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

imagingMode

An imaging mode (see below). See these constants:

`kQTVRStatic`

`kQTVRMotion`

`kQTVRAllModes`

imagingProperty

An imaging property (see below). See these constants:

`kQTVRImagingCorrection`

`kQTVRImagingQuality`

`kQTVRImagingDirectDraw`

`kQTVRImagingCurrentMode`

propertyValue

The desired value for the specified imaging property for the specified mode.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Default values for all imaging properties can be contained in a QuickTime VR movie file. If no defaults are specified in a movie file, the QuickTime VR Manager uses these values: for static mode, the `kQTVRImagingQuality` property is `codecHighQuality` and `kQTVRImagingDirectDraw` is TRUE; for motion mode, the `kQTVRImagingQuality` property is `codecLowQuality` and `kQTVRImagingDirectDraw` is TRUE. The default correction mode is `kQTVRFullCorrection` for both static and motion imaging modes. Note that it would look strange to have one correction mode for static imaging and a different correction mode for motion imaging. As a result, you should typically set the `imagingMode` parameter to `kQTVRAllModes` when setting a property of type `kQTVRImagingCorrection`.

Special Considerations

This function is valid only for panoramic nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

vrscript

vrscript.win

Declared In

`QuickTimeVR.h`

QTVRSetInteractionProperty

Sets the value of an interaction property.

```
OSErr QTVRSetInteractionProperty (
   QTVRInstance qtvr,
   UInt32 property,
   void *value
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

property

An interaction property type (see below). See these constants:

`kQTVRInteractionMouseClickHysteresis`

`kQTVRInteractionMouseClickTimeout`

`kQTVRInteractionPanTiltSpeed`

`kQTVRInteractionZoomSpeed`

`kQTVRInteractionTranslateOnMouseDown`

`kQTVRInteractionMouseMotionScale`

`kQTVRInteractionNudgeMode`

value

The desired value of the specified interaction property.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function sets the value of the interaction property of the type specified by the `property` parameter for the movie specified by the `qtvr` parameter to the value specified by the `value` parameter. For types that occupy 32 or fewer bits of memory, you pass the desired value itself (cast to a `void*` type) in the `value` parameter. For structures and floating-point values, you must pass a pointer to the desired value in the `value` parameter. Note that floating-point values are usually stored as 32-bit values, but compilers differ in how they pass floating-point values as parameters; as a result, this function demands that floating-point values always be passed by reference.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

vrscript

vrscript.win

Declared In

`QuickTimeVR.h`

QTVRSetLeavingNodeProc

Installs or removes a node-leaving procedure.

```
OSErr QTVRSetLeavingNodeProc (
   QTVRInstance qtvr,
   QTVRLeavingNodeUPP leavingNodeProc,
   SInt32 refCon,
   UInt32 flags
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

leavingNodeProc
: A Universal Procedure Pointer for a `QTVRLeavingNodeProc` callback. To remove a previously installed `QTVRLeavingNodeProc` callback, set `leavingNodeProc` to `NIL`.

refCon
: A reference constant. This value is passed to the specified node-leaving callback. Use this parameter to point to a data structure containing any information your callback needs.

flags
: Unused. Set this parameter to 0.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function installs the procedure specified by the `leavingNodeProc` parameter as a node-leaving procedure for the QuickTime VR movie specified by the `qtvr` parameter. Your procedure is called whenever a node is left (either in response to user actions or in response to QuickTime VR Manager functions that change nodes). The reference constant specified by the `refCon` parameter is passed unchanged to that node-leaving procedure.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
vrscript

vrscript.win

Declared In
`QuickTimeVR.h`

QTVRSetMouseDownTracking

Sets the state of mouse-down tracking.

```
OSErr QTVRSetMouseDownTracking (
   QTVRInstance qtvr,
   Boolean enable
);
```

Parameters

qtvr
: An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

enable
: A Boolean value that indicates whether QuickTime VR should handle mouse-down tracking for the specified movie (TRUE) or not (FALSE).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
By default, QuickTime VR tracks mouse clicks in a QuickTime VR movie and triggers hot spots as appropriate. If you disable mouse-down tracking (by passing FALSE in the `enable` parameter), you must call `QTVRMouseDown` (page 2023), `QTVRMouseStillDown` (page 2026), and `QTVRMouseUp` (page 2028) at the appropriate times to handle user actions.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeVR.h`

QTVRSetMouseOverHotSpotProc

Installs or removes a mouse over hot spot procedure.

```
OSErr QTVRSetMouseOverHotSpotProc (
   QTVRInstance qtvr,
   QTVRMouseOverHotSpotUPP mouseOverHotSpotProc,
   SInt32 refCon,
   UInt32 flags
);
```

Parameters

qtvr
An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

mouseOverHotSpotProc
A Universal Procedure Pointer for a `QTVRMouseOverHotSpotProc` callback. To remove a previously installed `QTVRMouseOverHotSpotUPP` callback, set `mouseOverHotSpotProc` to `NIL`.

refCon
A reference constant. This value is passed to the specified mouse over hot spot callback. Use this parameter to point to a data structure containing any information your callback needs.

flags
Unused. Set this parameter to 0.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function installs the routine specified by the `mouseOverHotSpotProc` parameter as a mouse over hot spot procedure for the QuickTime VR movie specified by the `qtvr` parameter. Subsequent user actions (such as moving the cursor over an enabled hot spot in that movie) cause the callback routine to be executed. The reference constant specified by the `refCon` parameter is passed unchanged to your callback routine.

Special Considerations

Your mouse over hot spot procedure is called only for enabled hot spots.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
vrcursors
vrcursors.win
vrscript
vrscript.win

Declared In
`QuickTimeVR.h`

QTVRSetMouseOverTracking

Sets the state of mouse-over tracking.

```
OSErr QTVRSetMouseOverTracking (
   QTVRInstance qtvr,
   Boolean enable
);
```

Parameters

qtvr
An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

enable
A Boolean value that indicates whether QuickTime VR should handle mouse-over tracking for the specified movie (TRUE) or not (FALSE).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
By default, QuickTime VR tracks mouse movements in a QuickTime VR movie and changes the shape of the cursor as appropriate. If you disable mouse-over tracking (by passing FALSE in the `enable` parameter), you must call `QTVRMouseEnter` (page 2025), `QTVRMouseWithin` (page 2031), and `QTVRMouseLeave` (page 2025) at the appropriate times to handle user actions.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeVR.h`

QTVRSetPanAngle

Sets the pan angle of a QuickTime VR movie.

```
OSErr QTVRSetPanAngle (
   QTVRInstance qtvr,
   float panAngle
);
```

Parameters

qtvr
: An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

panAngle
: The desired pan angle of the specified movie. This value is constrained by the maximum and minimum pan angles of the movie. If the angle falls outside of those constraints and the control setting `kQTVRWrapPan` is disabled, the angle is set to the minimum or maximum, whichever is closer. If wrapping is enabled, the pan angle is clipped to fall within the constraints. Pan angle values are also clipped if the requested pan angle, when combined with the current tilt angle and field of view, would cause an image to lie outside the current constraints.

Return Value
See `Error Codes`. Returns `noErr` if there is no error. This function returns the result code `constraintReachedErr` if wrapping is off and the angle is set to the minimum or maximum constraint value.

Special Considerations

The pan and tilt angles are subject to the current pan and tilt range constraints, as imposed by the viewing limits and the current field of view. Accordingly, if you want to change the field of view, you should do so before adjusting the pan or tilt angles. Otherwise, the pan and tilt angles are clipped against the current field of view, which may result in an incorrect view when you alter the field of view.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
vrmovies

vrmovies.win

vrscript

vrscript.win

vrspeech

Declared In
`QuickTimeVR.h`

QTVRSetPrescreenImagingCompleteProc

Installs or removes a prescreen buffer imaging completion procedure.

```
OSErr QTVRSetPrescreenImagingCompleteProc (
   QTVRInstance qtvr,
   QTVRImagingCompleteUPP imagingCompleteProc,
   SInt32 refCon,
   UInt32 flags
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

imagingCompleteProc

A Universal Procedure Pointer for a `QTVRImagingCompleteProc` callback. To remove a previously installed `QTVRImagingCompleteProc` callback, set `imagingCompleteProc` to `NIL`.

refCon

A reference constant. This value is passed to the specified prescreen buffer imaging completion callback. Use this parameter to point to a data structure containing any information your callback needs.

flags

A constant (see below) that causes a draw attempt on every idle passed to the movie controller besides being called whenever QuickTime VR finishes drawing an image into the prescreen buffer. See these constants:

`kQTVRPreScreenEveryIdle`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function installs the procedure specified by the `imagingCompleteProc` parameter as a prescreen buffer imaging completion procedure for the QuickTime VR movie specified by the `qtvr` parameter. Your procedure is called whenever QuickTime VR finishes drawing an image into the prescreen buffer. The reference constant specified by the `refCon` parameter is passed unchanged to that prescreen buffer imaging completion procedure.

Special Considerations

`QTVRSetPrescreenImagingCompleteProc` is valid only for panoramic nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

vrscript

vrscript.win

Declared In

`QuickTimeVR.h`

QTVRSetTiltAngle

Sets the tilt angle of a QuickTime VR movie.

```
OSErr QTVRSetTiltAngle (
   QTVRInstance qtvr,
   float tiltAngle
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

tiltAngle

The desired tilt angle of the specified movie. This value is constrained by the maximum and minimum tilt angles of the movie. If the angle falls outside of those constraints and the control setting `kQTVRWrapTilt` is disabled, the angle is set to the minimum or maximum, whichever is closer. If wrapping is enabled, the tilt angle is clipped to fall within the constraints. Tilt angle values are also clipped if the requested tilt angle, when combined with the current pan angle and field of view, would cause an image to lie outside the current constraints.

Return Value

See `Error Codes`. Returns `noErr` if there is no error. This function returns the result code `constraintReachedErr` if wrapping is off and the angle is set to the minimum or maximum constraint value.

Discussion

When a cylindrical panorama is zoomed all the way out (to its maximum vertical field of view), it can no longer be tilted because its entire vertical surface is exposed. Attempting to set the tilt angle will result in a `ConstraintReachedErr` error (except for the degenerate case of setting the tilt angle to its current value).

The tilt angle may not be zero when the panorama is fully zoomed out; it may be tilted by one line of pixels. The tilt angle is small in this case, typically 0.006, but its exact magnitude depends on the height of the panorama; the taller the panorama, the smaller the error.

Do not test for a tilt angle of exactly 0.0 or attempt to adjust the tilt angle of a fully zoomed-out panorama.

Special Considerations

The pan and tilt angles are subject to the current pan and tilt range constraints, as imposed by the viewing limits and the current field of view. Accordingly, if you want to change the field of view, you should do so before adjusting the pan or tilt angles. Otherwise, the pan and tilt angles are clipped against the current field of view, which may result in an incorrect view when you alter the field of view.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

vrmovies

vrmovies.win

vrscript

vrscript.win

vrspeech

Declared In

`QuickTimeVR.h`

QTVRSetTransitionProperty

Sets the value of a transition property.

```
OSErr QTVRSetTransitionProperty (
   QTVRInstance qtvr,
   UInt32 transitionType,
   UInt32 transitionProperty,
   SInt32 transitionValue
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

transitionType

A type of transition (see below). See these constants:

`kQTVRTransitionSwing`

transitionProperty

A type of transition property (see below). See these constants:

`kQTVRTransitionSpeed`

`kQTVRTransitionDirection`

transitionValue

The desired value for the specified transition property.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function sets the value of the transition property whose type is specified by the `transitionType` and `transitionProperty` parameters for the movie specified by the `qtvr` parameter to the value specified by the `transitionValue` parameter. Note that calling this function simply sets a transition property's value; you must still call `QTVREnableTransition` (page 1988) to enable that transition effect.

Special Considerations

`QTVRSetTransitionProperty` is valid only for panoramic nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

vrscript

vrscript.win

Declared In

`QuickTimeVR.h`

QTVRSetViewCenter

Sets the view center of a QuickTime VR movie.

```
OSErr QTVRSetViewCenter (
   QTVRInstance qtvr,
   const QTVRFloatPoint *viewCenter
);
```

Parameters

`qtvr`
: An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

`viewCenter`
: A pointer to a `QTVRFloatPoint` structure that contains the desired view center of the specified movie. This point is constrained by the current field of view of the movie. The values you pass in the `QTVRFloatPoint` structure are adjusted so that the magnified area does not show anything outside the view.

Return Value

See `Error Codes`. Returns `noErr` if there is no error. If the `kQTVRTranslation` control setting is disabled, this function returns the result code `constraintReachedErr` and doesn't change the current view center. You can use `QTVRSetControlSetting` (page 2042) to control the setting of `kQTVRTranslation`.

Special Considerations

`QTVRSetViewCenter` is valid only for object nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

vrscript

vrscript.win

Declared In

`QuickTimeVR.h`

QTVRSetViewCurrentTime

Sets the time in the current QTVR view.

```
OSErr QTVRSetViewCurrentTime (
   QTVRInstance qtvr,
   TimeValue time
);
```

Parameters

qtvr
: An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

time
: The desired time in the current view. This value should be greater than or equal to 0 and less than or equal to the value returned by `QTVRGetCurrentViewDuration` (page 1999).

Return Value

See `Error Codes`. Returns `noErr` if there is no error. This function returns the result code `timeNotInViewErr` if the specified time value is greater than or equal to the view duration of the specified object node; in addition, it sets the current view time to 1 less than the view duration. Similarly, this function returns the result code `timeNotInViewErr` if the specified time value is less than 0; in that case, it sets the current view time to 0.

Special Considerations

`QTVRSetViewCurrentTime` is valid only for object nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

vrscript

vrscript.win

Declared In

`QuickTimeVR.h`

QTVRSetViewParameter

Undocumented

```
OSErr QTVRSetViewParameter (
   QTVRInstance qtvr,
   UInt32 viewParameter,
   void *value,
   UInt32 flagsIn
);
```

Parameters

qtvr
: *Undocumented*

viewParameter
: *Undocumented*

value

Undocumented

flagsIn

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeVR.h`

QTVRSetViewRate

Sets the view rate of a QTVR object node.

```
OSErr QTVRSetViewRate (
   QTVRInstance qtvr,
   float rate
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

rate

The desired view rate of the specified movie. A view rate is a floating-point value in the range from -100.0 to +100.0. Positive values indicate forward rates, and negative values indicate reverse rates. Set this parameter to 0 to stop the movie.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function sets the view rate of the object node specified by the `qtvr` parameter to the rate specified by the rate parameter. A node's view rate might be modified by the current animation settings. If the value specified in the rate parameter lies outside the valid range, this function returns the result code `constraintReachedErr` and sets the view rate to the nearest constraint.

Special Considerations

`QTVRSetViewRate` is valid only for object nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code
vrscript

vrscript.win

Declared In
`QuickTimeVR.h`

QTVRSetViewState

Sets the value of a QTVR view state.

```
OSErr QTVRSetViewState (
   QTVRInstance qtvr,
   QTVRViewStateType viewStateType,
   UInt16 state
);
```

Parameters

`qtvr`
An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

`viewStateType`
A view state type (see below). See these constants:
`kQTVRDefault`
`kQTVRCurrent`
`kQTVRMouseDown`

`state`
The desired value of the specified type of view state.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Special Considerations
`QTVRSetViewState` is valid only for object nodes.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
vrscript

vrscript.win

Declared In
`QuickTimeVR.h`

QTVRSetVisible

Sets a VR movie's visibility state.

```
OSErr QTVRSetVisible (
   QTVRInstance qtvr,
   Boolean visible
);
```

Parameters

qtvr
: An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

visible
: A Boolean value that indicates whether the specified movie is to be visible (TRUE) or not (FALSE).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Setting the visibility state to FALSE is useful if you want to turn off imaging a QuickTime VR movie without purging the associated data from memory. When a panoramic node's visibility state is FALSE, the corrected image is still drawn to the prescreen buffer. You can access the data in that buffer by calling `QTVRSetPrescreenImagingCompleteProc` (page 2052).

Special Considerations

This function is valid only for panoramic nodes.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

vrscript

vrscript.win

Declared In

`QuickTimeVR.h`

QTVRShowDefaultView

Displays the default view of a QTVR node.

```
OSErr QTVRShowDefaultView (
   QTVRInstance qtvr
);
```

Parameters

qtvr
: An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function sets the default values of the pan angle, tilt angle, field of view, view center (for object nodes), default state, mouse-down state, and all applicable animation and control settings for the VR movie specified by the `qtvr` parameter. A VR movie's default view values are stored in the movie file.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
vrscript
vrscript.win

Declared In
`QuickTimeVR.h`

QTVRTiltToRow

Obtains the row number in the QTVR object image array that corresponds to a tilt angle.

```
short QTVRTiltToRow (
   QTVRInstance qtvr,
   float tiltAngle
);
```

Parameters

qtvr
An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

tiltAngle
A tilt angle.

Return Value
The zero-based row number in the current object image array that corresponds to the tilt angle specified by the `tiltAngle` parameter.

Special Considerations
This function is valid only for object nodes.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeVR.h`

QTVRTriggerHotSpot

Triggers a QTVR hot spot.

```
OSErr QTVRTriggerHotSpot (
   QTVRInstance qtvr,
   UInt32 hotSpotID,
   QTAtomContainer nodeInfo,
   QTAtom selectedAtom
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

hotSpotID

A hot spot ID.

nodeInfo

A node information atom container, obtained from a previous call to `QTVRGetNodeInfo` (page 2007). You can pass the `value` 0 in this parameter to have QuickTime determine the appropriate node information atom container.

selectedAtom

The atom of the hot spot to trigger. You can pass the `value` 0 in this parameter to have QuickTime determine the appropriate hot spot atom.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

One way you can use this function is to execute any hot spot without the user's having clicked it. Usually, you need only specify the `qtvr` instance and the hot spot ID. You can pass zero for the `nodeInfo` and `selectedAtom` parameters.

The more common use of this function is not in calling it directly, but in setting up an intercept procedure on it. This function is called internally by QuickTime whenever a user clicks a hot spot. You can intercept calls to trigger your custom hot spots, which allows you to perform any custom actions you desire.

When this function is called internally (and then intercepted by your intercept procedure), the `nodeInfo` and `selectedAtom` parameters have been properly set by QuickTime and are available for your use. For undefined hot spots that do not have an associated hot spot atom in the node info atom container, the `selectedAtom` parameter will be set to zero.

Special Considerations

You can call this function even on hot spots that are currently disabled.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

vrscript

vrscript.win

Declared In

`QuickTimeVR.h`

QTVRUpdate

Forces an immediate update of a QuickTime VR movie image.

```
OSErr QTVRUpdate (
   QTVRInstance qtvr,
   QTVRImagingMode imagingMode
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

imagingMode

An imaging mode. You can specify `kQTVRCurrentMode` (see below) to use the current imaging mode. See these constants:

`kQTVRCurrentMode`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function immediately updates the image for the QuickTime VR movie specified by the `qtvr` parameter, without waiting for the next call to `MoviesTask` (page 257) in your application's main event loop. If you plan to call this function repeatedly for a movie instance, then for improved performance you should bracket those calls with calls to `QTVRBeginUpdateStream` (page 1983) and `QTVREndUpdateStream` (page 1989).

Special Considerations

If you call this function after calling `QTVRBeginUpdateStream` (page 1983) but before calling `QTVREndUpdateStream` (page 1989), the `imagingMode` parameter passed to it must be the same as the `imagingMode` parameter passed to `QTVRBeginUpdateStream`. If you do not specify the same imaging mode to those two functions, an error will result.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

vrmovies

vrmovies.win

vrscript

vrscript.win

vrspeech

Declared In

`QuickTimeVR.h`

QTVRWrapAndConstrain

Preflights a change in the viewing or control characteristics of a QTVR object or panoramic node.

```
OSErr QTVRWrapAndConstrain (
   QTVRInstance qtvr,
   short kind,
   float value,
   float *result
);
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

kind

A constraint type (see below). See these constants:

`kQTVRPan`

`kQTVRTilt`

`kQTVRFieldOfView`

`kQTVRViewCenterH`

`kQTVRViewCenterV`

value

The desired value of the specified viewing characteristic.

result

On return, the value to which the specified viewing characteristic would be set if it were changed.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function returns, in the `result` parameter, the constrained or wrapped value that would result from setting the viewing or control characteristic specified by the `kind` parameter to the value specified by the `value` parameter. For example, if the `kind` parameter is set to `kQTVRPan`, then this function returns the value that would result from calling `QTVRSetPanAngle` (page 2051) with its `panAngle` parameter set to the `value` parameter. Similarly, you can use this function to find the current bounds of the view center. It takes into account the current constraints and wrapping modes of the node specified by the `qtvr` parameter. This function does not change the current view or other settings of the specified object or panorama.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

vrmovies

vrmovies.win

vrscript

vrscript.win

Declared In

`QuickTimeVR.h`

Callbacks

QTVRBackBufferImagingProc

An imaging procedure that draws directly into the back buffer for a QTVR panoramic node.

```
typedef OSErr (*QTVRBackBufferImagingProcPtr) (QTVRInstance qtvr, Rect *drawRect,
 UInt16 areaIndex, UInt32 flagsIn, UInt32 *flagsOut, SInt32 refCon);
```

If you name your function `MyQTVRBackBufferImagingProc`, you would declare it this way:

```
OSErr MyQTVRBackBufferImagingProc (
    QTVRInstance    qtvr,
    Rect            *drawRect,
    UInt16          areaIndex,
    UInt32          flagsIn,
    UInt32          *flagsOut,
    SInt32          refCon );
```

Parameters

qtvr
: An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

drawRect
: *Undocumented*

areaIndex
: *Undocumented*

flagsIn
: *Undocumented*

flagsOut
: *Undocumented*

refCon
: A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

Return Value
See `Error Codes`. Your callback should return `noErr` if there is no error.

Declared In
`QuickTimeVR.h`

QTVREnteringNodeProc

A routine called whenever a QTVR node is entered, in response either to user actions or QuickTime VR Manager functions that change nodes.

```
typedef OSErr (*QTVREnteringNodeProcPtr) (QTVRInstance qtvr, UInt32 nodeID, SInt32
 refCon);
```

If you name your function `MyQTVREnteringNodeProc`, you would declare it this way:

```
OSErr MyQTVREnteringNodeProc (
    QTVRInstance    qtvr,
    UInt32          nodeID,
    SInt32          refCon );
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

nodeID

A node ID. Set this parameter to `kQTVRCurrentNode` to designate the current node.

refCon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

Return Value

See `Error Codes`. Your callback should return `noErr` if there is no error.

Declared In

`QuickTimeVR.h`

QTVRImagingCompleteProc

An imaging completion procedure for a QuickTime VR movie, called whenever QTVR finishes drawing an image into the prescreen buffer.

```
typedef OSErr (*QTVRImagingCompleteProcPtr) (QTVRInstance qtvr, SInt32 refCon);
```

If you name your function `MyQTVRImagingCompleteProc`, you would declare it this way:

```
OSErr MyQTVRImagingCompleteProc (
    QTVRInstance    qtvr,
    SInt32          refCon );
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

refCon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

Return Value

See `Error Codes`. Your callback should return `noErr` if there is no error.

Declared In

`QuickTimeVR.h`

QTVRInterceptProc

A routine that is called when certain QTVR functions are intercepted.

```
typedef void (*QTVRInterceptProcPtr) (QTVRInstance qtvr, QTVRInterceptPtr qtvrMsg,
 SInt32 refCon, Boolean *cancel);
```

If you name your function `MyQTVRInterceptProc`, you would declare it this way:

```
void MyQTVRInterceptProc (
    QTVRInstance        qtvr,
    QTVRInterceptPtr    qtvrMsg,
    SInt32              refCon,
    Boolean             *cancel );
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

qtvrMsg

A pointer to a `QTVRInterceptRecord` structure that determines which functions are intercepted.

refCon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

cancel

A pointer to a Boolean; set to TRUE if the intercept is cancelled.

Declared In

`QuickTimeVR.h`

QTVRLeavingNodeProc

A routine called whenever a QTVR node is left, in response either to user actions or QuickTime VR Manager functions that change nodes.

```
typedef OSErr (*QTVRLeavingNodeProcPtr) (QTVRInstance qtvr, UInt32 fromNodeID,
UInt32 toNodeID, Boolean *cancel, SInt32 refCon);
```

If you name your function `MyQTVRLeavingNodeProc`, you would declare it this way:

```
OSErr MyQTVRLeavingNodeProc (
    QTVRInstance    qtvr,
    UInt32          fromNodeID,
    UInt32          toNodeID,
    Boolean         *cancel,
    SInt32          refCon );
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

fromNodeID

The ID of the node being left. Set this parameter to `kQTVRCurrentNode` to designate the current node.

toNodeID

The ID of the node being entered. Set this parameter to `kQTVRCurrentNode` to designate the current node.

cancel

A pointer to a Boolean; set to TRUE if the callback is cancelled.

refCon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

Return Value

See `Error Codes`. Your callback should return `noErr` if there is no error.

Declared In

`QuickTimeVR.h`

QTVRMouseOverHotSpotProc

A routine that is called when the mouse is over a hot spot in a QTVR movie.

```
typedef OSErr (*QTVRMouseOverHotSpotProcPtr) (QTVRInstance qtvr, UInt32 hotSpotID,
 UInt32 flags, SInt32 refCon);
```

If you name your function `MyQTVRMouseOverHotSpotProc`, you would declare it this way:

```
OSErr MyQTVRMouseOverHotSpotProc (
    QTVRInstance    qtvr,
    UInt32          hotSpotID,
    UInt32          flags,
    SInt32          refCon );
```

Parameters

qtvr

An instance of a QuickTime VR movie. You can get this value by calling `QTVRGetQTVRInstance` (page 2009).

hotSpotID

A hot spot ID.

flags

Undocumented

refCon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

Return Value

See `Error Codes`. Your callback should return `noErr` if there is no error.

Declared In

`QuickTimeVR.h`

Data Types

QTVRAngularUnits

Represents a type used by the Virtual Reality API.

```
typedef UInt32 QTVRAngularUnits;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeVR.h`

QTVRAreaOfInterest

Contains information passed to QTVRSetBackBufferImagingProc.

```
struct QTVRAreaOfInterest {
     float     panAngle;
     float     tiltAngle;
     float     width;
     float     height;
     UInt32    flags;
 };
```

Fields
`panAngle`
Discussion
The pan angle of the upper-left coordinate (in panorama space) of the area of interest.

`tiltAngle`
Discussion
The tilt angle of the upper-left coordinate (in panorama space) of the area of interest.

`width`
Discussion
The width of the area of interest.

`height`
Discussion
The height of the area of interest.

`flags`
Discussion
A set of bit flags (see below) that indicate when to call the back buffer imaging procedure for this area of interest. See these constants:
`kQTVRBackBufferEveryUpdate`
`kQTVRBackBufferEveryIdle`
`kQTVRBackBufferAlwaysRefresh`

Discussion

The `areasOfInterest` parameter to `QTVRSetBackBufferImagingProc` (page 2039) specifies an array of `QTVRAreaOfInterest` structures, each one of which indicates a rectangular area in the QTVR back buffer.

Related Functions

`QTVRSetBackBufferImagingProc` (page 2039)

Declared In

`QuickTimeVR.h`

QTVRBackBufferImagingUPP

Represents a type used by the Virtual Reality API.

```
typedef STACK_UPP_TYPE(QTVRBackBufferImagingProcPtr) QTVRBackBufferImagingUPP;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeVR.h`

QTVRControlSetting

Represents a type used by the Virtual Reality API.

```
typedef UInt32 QTVRControlSetting;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeVR.h`

QTVRCursorRecord

Contains information passed to QTVRReplaceCursor.

```
struct QTVRCursorRecord {
     UInt16    theType;
     SInt16    rsrcID;
     Handle    handle;
 };
```

Fields

`theType`

Discussion

A constant (see below) that defines the type of cursor to replace. See these constants:

`kQTVRUseDefaultCursor`

`kQTVRStdCursorType`

`kQTVRColorCursorType`

`rsrcID`

Discussion

The resource ID of the cursor to replace; see `QTVR Cursors`.

`handle`

Discussion

A handle to the cursor data that is to replace the specified cursor. If `theType` is `kQTVRUseDefaultCursor`, then this field should contain `NIL`.

Discussion

The `cursRecord` parameter to `QTVRReplaceCursor` (page 2035) specifies a `QTVRCursorRecord` structure, which indicates the cursor to replace and its replacement cursor.

Version Notes

In earlier versions of QuickTime, `theType` was called the `type` field.

Related Functions

`QTVRReplaceCursor` (page 2035)

Declared In

`QuickTimeVR.h`

QTVREnteringNodeUPP

Represents a type used by the Virtual Reality API.

```
typedef STACK_UPP_TYPE(QTVREnteringNodeProcPtr) QTVREnteringNodeUPP;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeVR.h`

QTVRFloatPoint

Specifies a point in a QTVR panorama or object.

```
struct QTVRFloatPoint {
     float    x;
     float    y;
 };
```

Fields

`x`

Discussion

The horizontal coordinate of the point.

`y`

Discussion

The vertical coordinate of the point.

Related Functions
QTVRAnglesToCoord (page 1982)
QTVRCoordToAngles (page 1985)
QTVRGetViewCenter (page 2013)

Declared In
QuickTimeVR.h

QTVRImagingCompleteUPP

Represents a type used by the Virtual Reality API.

```
typedef STACK_UPP_TYPE(QTVRImagingCompleteProcPtr) QTVRImagingCompleteUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
QuickTimeVR.h

QTVRImagingMode

Represents a type used by the Virtual Reality API.

```
typedef UInt32 QTVRImagingMode;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
QuickTimeVR.h

QTVRInstance

Represents a type used by the Virtual Reality API.

```
typedef struct OpaqueQTVRInstance * QTVRInstance;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
QuickTimeVR.h

QTVRInterceptRecord

Contains information about a QTVRInterceptProc being intercepted by QTVR.

```
struct QTVRInterceptRecord {
     SInt32    reserved1;
     SInt32    selector;
     SInt32    reserved2;
     SInt32    reserved3;
     SInt32    paramCount;
     void *    parameter[6];
 };
```

Fields

`reserved1`

Discussion

Reserved; do not use.

`selector`

Discussion

A constant that indicates which routine has triggered your intercept procedure (see below). See these constants:

- `kQTVRSetPanAngleSelector`
- `kQTVRSetTiltAngleSelector`
- `kQTVRSetFieldOfViewSelector`
- `kQTVRSetViewCenterSelector`
- `kQTVRMouseEnterSelector`
- `kQTVRMouseWithinSelector`
- `kQTVRMouseLeaveSelector`
- `kQTVRMouseDownSelector`
- `kQTVRMouseStillDownSelector`
- `kQTVRMouseUpSelector`
- `kQTVRTriggerHotSpotSelector`
- `kQTVRGetHotSpotTypeSelector`

`reserved2`

Discussion

Reserved; do not use.

`reserved3`

Discussion

Reserved; do not use.

`paramCount`

Discussion

The number of items in the `parameter` field.

`parameter`

Discussion

An array that holds, in order, the parameters that were passed to the intercepted function, minus the QTVR instance parameter. For example, if you intercept the `QTVRSetPanAngle` function, the `parameter` field contains a single item, a pointer to a floating-point value that is the new pan angle. You can determine how many items the `parameter` field contains by inspecting the `paramCount` field.

Related Functions

`QTVRCallInterceptedProc` (page 1984)

Declared In
`QuickTimeVR.h`

QTVRInterceptUPP

Represents a type used by the Virtual Reality API.

```
typedef STACK_UPP_TYPE(QTVRInterceptProcPtr) QTVRInterceptUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeVR.h`

QTVRLeavingNodeUPP

Represents a type used by the Virtual Reality API.

```
typedef STACK_UPP_TYPE(QTVRLeavingNodeProcPtr) QTVRLeavingNodeUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeVR.h`

QTVRMouseOverHotSpotUPP

Represents a type used by the Virtual Reality API.

```
typedef STACK_UPP_TYPE(QTVRMouseOverHotSpotProcPtr) QTVRMouseOverHotSpotUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeVR.h`

QTVRNudgeControl

Represents a type used by the Virtual Reality API.

```
typedef UInt32 QTVRNudgeControl;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeVR.h`

QTVRObjectAnimationSetting

Represents a type used by the Virtual Reality API.

```
typedef UInt32 QTVRObjectAnimationSetting;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeVR.h`

QTVRProcSelector

Represents a type used by the Virtual Reality API.

```
typedef UInt32 QTVRProcSelector;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeVR.h`

QTVRViewStateType

Represents a type used by the Virtual Reality API.

```
typedef UInt32 QTVRViewStateType;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeVR.h`

Constants

kQTVRBackBufferAlwaysRefresh

Constants grouped with kQTVRBackBufferAlwaysRefresh.

```
enum {
  kQTVRBackBufferEveryUpdate    = 1L << 0,
  kQTVRBackBufferEveryIdle      = 1L << 1,
  kQTVRBackBufferAlwaysRefresh  = 1L << 2,
  kQTVRBackBufferHorizontal     = 1L << 3 /* Requires that backbuffer proc be
long-rowBytes aware (gestaltQDHasLongRowBytes)*/
};
```

Constants

`kQTVRBackBufferEveryUpdate`

If this bit is set, the back buffer imaging procedure is to be called whenever QuickTime is about to update the window containing the specified QuickTime VR movie instance. That is, the procedure is called just before QuickTime unwarps the back buffer image into the prescreen buffer and redraws the screen image.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeVR.h`.

`kQTVRBackBufferEveryIdle`

If this bit is set, the back buffer imaging procedure is to be called when either `MCIsPlayerEvent` (page 1204) is called with the idle parameter, or when `MCIdle` (page 1202) is called. Its purpose is to cause the software to draw as often as possible.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeVR.h`.

`kQTVRBackBufferAlwaysRefresh`

If this bit is set, the back buffer is always refreshed to the proper movie data just before your back buffer imaging procedure is called. If your back buffer imaging procedure completely overwrites the rectangle passed to it, you should not set this bit.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeVR.h`.

Declared In

`QuickTimeVR.h`

QTVRGoToNodeID Values

Constants passed to QTVRGoToNodeID.

```
enum {
  kQTVRCurrentNode              = 0,
  kQTVRPreviousNode             = (long)0x80000000,
  kQTVRDefaultNode              = (long)0x80000001
};
```

Declared In

`QuickTimeVR.h`

QTVRSetViewState Values

Constants passed to QTVRSetViewState.

```
enum {
  kQTVRDefault                 = 0,
  kQTVRCurrent                 = 2,
  kQTVRMouseDown               = 3
};
```

Declared In
`QuickTimeVR.h`

QTVRSetBackBufferPrefs Values

Constants passed to QTVRSetBackBufferPrefs.

```
enum {
  kQTVRDefaultRes              = 0,
  kQTVRFullRes                 = 1L << 0,
  kQTVRHalfRes                 = 1L << 1,
  kQTVRQuarterRes              = 1L << 2
};
enum {
  kQTVRMinimumCache            = -1,
  kQTVRSuggestedCache          = 0,
  kQTVRFullCache               = 1
};
```

Constants

`kQTVRQuarterRes`
: One-quarter the full resolution of the image.
: Available in Mac OS X v10.0 and later.
: Declared in `QuickTimeVR.h`.

`kQTVRMinimumCache`
: The minimum cache size required to display the specified VR movie.
: Available in Mac OS X v10.0 and later.
: Declared in `QuickTimeVR.h`.

`kQTVRSuggestedCache`
: The suggested cache size, a cache large enough to allow full zooming out of the panorama.
: Available in Mac OS X v10.0 and later.
: Declared in `QuickTimeVR.h`.

Declared In
`QuickTimeVR.h`

QTVRSetAngularUnits Values

Constants passed to QTVRSetAngularUnits.

```
enum {
  kQTVRDegrees                  = 0,
  kQTVRRadians                  = 1
};
```

Declared In
`QuickTimeVR.h`

QTVREnableHotSpot Values

Constants passed to QTVREnableHotSpot.

```
enum {
  kQTVRHotSpotID                = 0,
  kQTVRHotSpotType              = 1,
  kQTVRAllHotSpots              = 2
};
```

Declared In
`QuickTimeVR.h`

kQTVRImagingCorrection

Constants grouped with kQTVRImagingCorrection.

```
enum {
  kQTVRImagingCorrection        = 1,
  kQTVRImagingQuality           = 2,
  kQTVRImagingDirectDraw        = 3,
  kQTVRImagingCurrentMode       = 100   /* Get Only*/
};
```

Declared In
`QuickTimeVR.h`

QTVRSetInteractionProperty Values

Constants passed to QTVRSetInteractionProperty.

```
enum {
  kQTVRInteractionMouseClickHysteresis = 1, /* pixels within which the mouse is
considered not to have moved (UInt16)*/
  kQTVRInteractionMouseClickTimeout = 2, /* ticks after which a mouse click times
 out and turns into panning (UInt32)*/
  kQTVRInteractionPanTiltSpeed  = 3,    /* control the relative pan/tilt speed from
 1 (slowest) to 10 (fastest). (UInt32) Default is 5;*/
  kQTVRInteractionZoomSpeed     = 4,    /* control the relative zooming speed from
 1 (slowest) to 10 (fastest). (UInt32) Default is 5;*/
  kQTVRInteractionTranslateOnMouseDown = 101, /* Holding MouseDown with this setting
 translates zoomed object movies (Boolean)*/
  kQTVRInteractionMouseMotionScale = 102, /* The maximum angle of rotation caused
 by dragging across the display window. (* float)*/
  kQTVRInteractionNudgeMode     = 103   /* A QTVRNudgeMode: rotate, translate, or
 the same as the current mouse mode. Requires QTVR 2.1*/
};
```

Declared In
`QuickTimeVR.h`

kQTVRDontLoopViewFrames

Constants grouped with kQTVRDontLoopViewFrames.

```
enum {
                                        /* View Frame Animation Settings*/
  kQTVRPalindromeViewFrames     = 1,
  kQTVRStartFirstViewFrame      = 2,
  kQTVRDontLoopViewFrames       = 3,
  kQTVRPlayEveryViewFrame      = 4,    /* Requires QTVR 2.1 (kQTVRAPIMajorVersion02
 + kQTVRAPIMinorVersion10)*/
                                        /* View Animation Settings*/
  kQTVRSyncViewToFrameRate      = 16,
  kQTVRPalindromeViews          = 17,
  kQTVRPlayStreamingViews      = 18    /* Requires QTVR 2.1 (kQTVRAPIMajorVersion02
 + kQTVRAPIMinorVersion10)*/
};
```

Declared In
`QuickTimeVR.h`

QTVRWrapAndConstrain Values

Constants passed to QTVRWrapAndConstrain.

```
enum {
  kQTVRPan                      = 0,
  kQTVRTilt                     = 1,
  kQTVRFieldOfView              = 2,
  kQTVRViewCenterH              = 4,    /* WrapAndConstrain only*/
  kQTVRViewCenterV              = 5     /* WrapAndConstrain only*/
};
```

Declared In
`QuickTimeVR.h`

QTVRSetPrescreenImagingCompleteProc Values

Constants passed to QTVRSetPrescreenImagingCompleteProc.

```
enum {
  kQTVRPreScreenEveryIdle       = 1L << 0 /* Requires QTVR 2.1
(kQTVRAPIMajorVersion02 + kQTVRAPIMinorVersion10)*/
};
```

Declared In
QuickTimeVR.h

kQTVRDown

Constants grouped with kQTVRDown.

```
enum {
  kQTVRRight                    = 0,
  kQTVRUpRight                  = 45,
  kQTVRUp                       = 90,
  kQTVRUpLeft                   = 135,
  kQTVRLeft                     = 180,
  kQTVRDownLeft                 = 225,
  kQTVRDown                     = 270,
  kQTVRDownRight                = 315
};
```

Constants

`kQTVRLeft`

Swing the view to the left.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeVR.h`.

`kQTVRDown`

Swing the view down.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeVR.h`.

Declared In
QuickTimeVR.h

kQTVRGetHotSpotTypeSelector

Constants grouped with kQTVRGetHotSpotTypeSelector.

```
enum {
  kQTVRSetPanAngleSelector      = 0x2000,
  kQTVRSetTiltAngleSelector     = 0x2001,
  kQTVRSetFieldOfViewSelector   = 0x2002,
  kQTVRSetViewCenterSelector    = 0x2003,
  kQTVRMouseEnterSelector       = 0x2004,
  kQTVRMouseWithinSelector      = 0x2005,
  kQTVRMouseLeaveSelector       = 0x2006,
  kQTVRMouseDownSelector        = 0x2007,
  kQTVRMouseStillDownSelector   = 0x2008,
  kQTVRMouseUpSelector          = 0x2009,
  kQTVRTriggerHotSpotSelector   = 0x200A,
  kQTVRGetHotSpotTypeSelector   = 0x200B, /* Requires QTVR 2.1
(kQTVRAPIMajorVersion02 + kQTVRAPIMinorVersion10)*/
  kQTVRSetViewParameterSelector = 0x200C, /* Requires QTVR 5.0
(kQTVRAPIMajorVersion05 + kQTVRAPIMinorVersion00)*/
 kQTVRGetViewParameterSelector = 0x200D /* Requires QTVR 5.0 (kQTVRAPIMajorVersion05
 + kQTVRAPIMinorVersion00)*/
};
```

Constants

`kQTVRSetPanAngleSelector`

Value is 0x2000.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeVR.h`.

`kQTVRSetTiltAngleSelector`

Value is 0x2000.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeVR.h`.

`kQTVRSetFieldOfViewSelector`

Value is 0x2002.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeVR.h`.

`kQTVRSetViewCenterSelector`

Value is 0x2003.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeVR.h`.

`kQTVRMouseEnterSelector`

Value is 0x2004.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeVR.h`.

`kQTVRMouseWithinSelector`

Value is 0x2005.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeVR.h`.

`kQTVRMouseLeaveSelector`
Value is 0x2006.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeVR.h`.

`kQTVRMouseDownSelector`
Value is 0x2007.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeVR.h`.

`kQTVRMouseStillDownSelector`
Value is 0x2008.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeVR.h`.

`kQTVRMouseUpSelector`
Value is 0x2009.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeVR.h`.

`kQTVRTriggerHotSpotSelector`
Value is 0x200A.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeVR.h`.

`kQTVRGetHotSpotTypeSelector`
Value is 0x200B.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeVR.h`.

Declared In
`QuickTimeVR.h`

kQTVRAllModes

Constants grouped with kQTVRAllModes.

```
enum {
  kQTVRStatic                  = 1,
  kQTVRMotion                  = 2,
  kQTVRCurrentMode             = 0,    /* Special Value for QTVRUpdate*/
  kQTVRAllModes                = 100   /* Special value for QTVRSetProperty*/
};
```

Declared In
`QuickTimeVR.h`

QTVRSetTransitionProperty Values

Constants passed to QTVRSetTransitionProperty.

```
enum {
  kQTVRTransitionSpeed          = 1,
  kQTVRTransitionDirection      = 2
};
enum {
  kQTVRTransitionSwing          = 1
};
```

Declared In
`QuickTimeVR.h`

QTVRCursorRecord Values

Constants passed to QTVRCursorRecord.

```
enum {
  kQTVRUseDefaultCursor         = 0,
  kQTVRStdCursorType            = 1,
  kQTVRColorCursorType          = 2
};
```

Constants

`kQTVRUseDefaultCursor`
Restore the default cursor. In this case, the `handle` field of the cursor record should contain `NIL`.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeVR.h`.

`kQTVRStdCursorType`
The cursor is a standard black-and-white cursor.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeVR.h`.

Declared In
`QuickTimeVR.h`

kQTVRCube

Constants grouped with kQTVRCube.

```
enum {
  kQTVRUseMovieGeometry         = 0,
  kQTVRVerticalCylinder         = 'vcyl',
  kQTVRHorizontalCylinder       = 'hcyl',
  kQTVRCube                     = 'cube'
};
```

Declared In
`QuickTimeVR.h`

QTVRSetControlSetting Values

Constants passed to QTVRSetControlSetting.

```
enum {
  kQTVRWrapPan                  = 1,
  kQTVRWrapTilt                 = 2,
  kQTVRCanZoom                  = 3,
  kQTVRReverseHControl          = 4,
  kQTVRReverseVControl          = 5,
  kQTVRSwapHVControl            = 6,
  kQTVRTranslation              = 7
};
```

Declared In

`QuickTimeVR.h`

Sequence Grabber Reference for QuickTime

Framework:	Frameworks/QuickTime.framework
Declared in	QuickTimeComponents.h

Overview

Sequence Grabber components allow applications to obtain digitized data from external sources, such as video capture boards. The digitized data can be previewed, saved as a QuickTime movie, or both. Sequence grabber components allow applications to capture audio and video easily, without concern for the details of how the data is acquired.

Functions by Task

Configuration Functions for All Channel Components

`SGGetChannelDeviceAndInputNames` (page 2127)
Returns the sequence grabber's current device and input names.

`SGGetChannelRefCon` (page 2131)
Returns a reference constant that was previously set by SGSetChannelRefCon.

`SGGetDataRate` (page 2139)
Determines for a sequence grabber how much recording time is left.

Configuration Functions for Video Channel Components

`SGAlignChannelRect` (page 2112)
Determines whether or not a channel prefers to draw at a particular screen location.

Configuring Sequence Grabber Channel Components

`SGInitChannel` (page 2167)
Initializes a channel component.

Configuring Sequence Grabber Components

`SGDisposeChannel` (page 2121)
: Removes a channel from a sequence grabber component.

`SGGetAlignmentProc` (page 2124)
: Obtains information about the best screen positions for a sequence grabber's video image in terms of appearance and maximum performance.

`SGGetDataOutput` (page 2137)
: Determines the movie file that is currently assigned to a sequence grabber component and the control flags that would govern a record operation.

`SGGetDataRef` (page 2139)
: Determines the data reference currently assigned to a sequence grabber component and the control flags that would govern a record operation.

`SGGetGWorld` (page 2142)
: Determines the graphics port and device for a sequence grabber component.

`SGGetIndChannel` (page 2142)
: Collects information about all of the channel components currently in use by a sequence grabber component.

`SGInitialize` (page 2168)
: Initializes the sequence grabber component.

`SGNewChannel` (page 2169)
: Creates a sequence grabber channel and assigns a channel component to the channel.

`SGNewChannelFromComponent` (page 2170)
: Creates a sequence grabber channel and assigns a channel component to the channel.

`SGSetDataOutput` (page 2197)
: Specifies the movie file and options for a sequence grabber record operation.

`SGSetDataProc` (page 2198)
: Specifies a data function for use by the sequence grabber.

`SGSetDataRef` (page 2199)
: Specifies the destination data reference for a record operation.

`SGSetGWorld` (page 2202)
: Establishes the graphics port and device for a sequence grabber component.

Controlling Sequence Grabber Channel Components

`SGWriteSamples` (page 2227)
: Called by a sequence grabber component when it is ready to add recorded data to a movie.

Controlling Sequence Grabber Components

`SGGetLastMovieResID` (page 2144)
: Retrieves the last resource ID used by the sequence grabber component.

`SGGetMode` (page 2145)
: Determines whether a sequence grabber component is in preview mode or record mode.

`SGGetMovie` (page 2145)
: Returns a reference to the movie that contains the data collected during a record operation.

`SGGetPause` (page 2149)
: Determines whether the sequence grabber is paused.

`SGGrabPict` (page 2165)
: Lets your application obtain a Picture structure from a sequence grabber component.

`SGIdle` (page 2166)
: Provides processing time for sequence grabber components.

`SGPause` (page 2184)
: Suspends or restarts a sequence grabber record or preview operation.

`SGPrepare` (page 2185)
: Instructs a sequence grabber to get ready to begin a preview or record operation.

`SGRelease` (page 2186)
: Instructs the sequence grabber to release any system resources it allocated when you called SGPrepare.

`SGStartPreview` (page 2222)
: Instructs the sequence grabber to begin processing data from its channels.

`SGStartRecord` (page 2222)
: Instructs the sequence grabber component to begin collecting data from its channels.

`SGStop` (page 2223)
: Stops a preview or record operation.

`SGUpdate` (page 2224)
: Informs your component about update events, to update its display.

Managing Your Panel Component

`SGPanelCanRun` (page 2172)
: Lets a sequence grabber component determine whether a panel component can work with the current sequence grabber channel component.

`SGPanelGetDitl` (page 2174)
: Lets a sequence grabber component determine the dialog items managed by your panel component.

`SGPanelGetDITLForSize` (page 2175)
: Returns user interface elements that fit within a specified size panel.

`SGPanelGetTitle` (page 2177)
: Gets the displayed title of a sequence grabber panel.

`SGPanelInstall` (page 2177)
: Installs added items in a sequence grabber settings dialog box before the dialog box is displayed to the user.

`SGPanelRemove` (page 2179)
: Removes a panel from the sequence grabber settings dialog box.

`SGPanelSetGrabber` (page 2181)
: Identifies a sequence grabber component to a panel component.

`SGPanelSetResFile` (page 2182)
: Lets the sequence grabber pass a resource file's reference number.

Managing Your Panel's Settings

`SGPanelGetSettings` (page 2176)
Retrieves a panel's current settings for a sequence grabber component.

`SGPanelSetSettings` (page 2182)
Restores a panel's current settings for a sequence grabber component.

Processing Your Panel's Events

`SGPanelEvent` (page 2173)
Lets a component receive and process dialog events.

`SGPanelItem` (page 2178)
Receives and processes mouse clicks in the sequence grabber settings dialog box.

`SGPanelSetEventFilter` (page 2180)
Sets the event filter callback for a sequence grabber panel component.

`SGPanelValidateInput` (page 2183)
Validates the contents of the user dialog box for a sequence grabber component.

Text Channel Support

`SGGetTextReturnToSpaceValue` (page 2156)
Indicates whether the text channel component should replace return characters with spaces.

`SGSetFontName` (page 2200)
Sets the name of the font to be used to display text for a text channel component.

`SGSetFontSize` (page 2201)
Sets the font size to be used to display text for a text channel component.

`SGSetJustification` (page 2204)
Sets the alignment to be used to display text for a text channel component.

`SGSetTextBackColor` (page 2212)
Sets the background color to be used for the text box.

`SGSetTextForeColor` (page 2213)
Sets the color to be used to display text.

`SGSetTextReturnToSpaceValue` (page 2213)
Determines whether the text channel component should replace return characters with spaces.

Utility Functions for Sequence Grabber Channel Components

`SGAddExtendedFrameReference` (page 2107)
Stores extended sample references for a channel component.

`SGAddExtendedMovieData` (page 2107)
Adds data to a movie without writing data to a movie file.

`SGAddFrameReference` (page 2110)
Stores sample references for a channel component.

`SGAddMovieData` (page 2110)
  Lets a channel component add data to a movie.

`SGAddOutputDataRefToMedia` (page 2111)
  Manages capture sessions that involve multiple data references.

`SGChangedSource` (page 2113)
  Informs the sequence grabber that a component is now using a different device.

`SGChannelGetDataSourceName` (page 2115)
  Returns the data source name for a track.

`SGChannelGetRequestedDataRate` (page 2115)
  Returns the current maximum data rate requested for a channel.

`SGChannelSetDataSourceName` (page 2117)
  Sets the data source name for a track.

`SGChannelSetRequestedDataRate` (page 2118)
  Specifies the maximum requested data rate for a channel.

`SGGetAdditionalSoundRates` (page 2123)
  Returns the additional sound sample rates added to a specified sequence grabber sound channel.

`SGGetNextExtendedFrameReference` (page 2146)
  Allows a channel component to retrieve the sample references stored previously by SGAddExtendedMovieData or SGAddExtendedFrameReference.

`SGGetNextFrameReference` (page 2147)
  Lets a channel component retrieve the sample references that were stored by calling SGAddMovieData or SGAddFrameReference.

`SGGetPreferredPacketSize` (page 2150)
  Returns the preferred packet size for the sequence grabber component.

`SGGetUserVideoCompressorList` (page 2157)
  Returns the video compression formats to be displayed by the specified sequence grabber video channel.

`SGSetAdditionalSoundRates` (page 2186)
  Specifies a list of sound sample rates to be included in the sequence grabber's sound settings dialog box.

`SGSetPreferredPacketSize` (page 2208)
  Sets the preferred packet size for the sequence grabber channel component.

`SGSetUserVideoCompressorList` (page 2215)
  Specifies the list of video compression formats to be included in the sequence grabber's video settings dialog box.

`SGSortDeviceList` (page 2221)
  Sorts a device list alphabetically.

`SGWriteExtendedMovieData` (page 2226)
  Allows your channel component to add data to a movie.

`SGWriteMovieData` (page 2227)
  Lets a channel component add data to a movie.

Video Channel Callback Functions

`SGGetVideoBottlenecks` (page 2159)
: Determines the callback functions that have been assigned to a video channel.

`SGSetVideoBottlenecks` (page 2217)
: Assigns callback functions to a video channel.

Working With Channel Characteristics

`SGGetChannelBounds` (page 2126)
: Determines a channel's display boundary rectangle.

`SGGetChannelClip` (page 2126)
: Retrieves a channel's clipping region.

`SGGetChannelInfo` (page 2128)
: Determines how a channel's data is represented to the user: as visual data or audio data, or both.

`SGGetChannelMatrix` (page 2129)
: Retrieves a channel's display transformation matrix.

`SGGetChannelMaxFrames` (page 2130)
: Determines the number of frames left to be captured from a specified channel.

`SGGetChannelPlayFlags` (page 2130)
: Retrieves the playback control flags that you set with SGSetChannelPlayFlags.

`SGGetChannelSampleDescription` (page 2132)
: Retrieves a channel's sample description structure.

`SGGetChannelTimeScale` (page 2134)
: Lets the sequence grabber retrieve a channel's time scale.

`SGGetChannelUsage` (page 2135)
: Determines how the sequence grabber component is using a channel.

`SGGetChannelVolume` (page 2135)
: Determines a channel's sound volume setting.

`SGSetChannelBounds` (page 2187)
: Specifies a channel's display boundary rectangle.

`SGSetChannelClip` (page 2188)
: Sets a channel's clipping region.

`SGSetChannelMatrix` (page 2190)
: Sets a channel's display transformation matrix.

`SGSetChannelMaxFrames` (page 2190)
: Limits the number of frames that the sequence grabber will capture from a specified channel.

`SGSetChannelPlayFlags` (page 2192)
: Adjusts the speed and quality with which the sequence grabber displays data from a channel.

`SGSetChannelRefCon` (page 2192)
: Sets the value of a reference constant that is passed to your callback functions for channel components.

`SGSetChannelUsage` (page 2195)
: Specifies how a channel is to be used by the sequence grabber component.

`SGSetChannelVolume` (page 2195)
Sets a channel's sound volume.

Working With Channel Devices

`SGAppendDeviceListToMenu` (page 2113)
Places a list of device names into a specified menu.

`SGDisposeDeviceList` (page 2122)
Disposes of a device list.

`SGGetChannelDeviceList` (page 2128)
Retrieves a list of the devices that are valid for a specified channel.

`SGSetChannelDevice` (page 2188)
Assigns a device to a channel.

`SGSetChannelSettingsStateChanging` (page 2194)
Tells a sequence grabber channel of the beginning and end of a group of setting calls.

`SGSetSettingsSummary` (page 2209)
Sets the summary of sequence grabber settings that is displayed in the lower left corner of the sequence grabber dialog.

Working With Sequence Grabber Characteristics

`SGGetFlags` (page 2140)
Retrieves a sequence grabber's control flags.

`SGGetMaximumRecordTime` (page 2144)
Determines the time limit you have set for a record operation.

`SGGetStorageSpaceRemaining` (page 2154)
Monitors the amount of space remaining for use during a record operation.

`SGGetTimeBase` (page 2156)
Retrieves a reference to the time base that is being used by a sequence grabber component.

`SGGetTimeRemaining` (page 2157)
Obtains an estimate of the amount of recording time that remains for the current record operation.

`SGSetFlags` (page 2200)
Passes control information about the current operation to the sequence grabber component.

`SGSetMaximumRecordTime` (page 2205)
Limits the duration of a record operation

Working With Sequence Grabber Outputs

`SGDisposeOutput` (page 2123)
Disposes of an existing sequence grabber output.

`SGGetDataOutputStorageSpaceRemaining` (page 2137)
Returns the amount of space remaining in the data reference associated with an output.

`SGGetOutputDataReference` (page 2147)
: Returns information about the data reference associated with the specified sequence grabber output.

`SGGetOutputMaximumOffset` (page 2148)
: Returns the maximum offset for data written to the specified sequence grabber output.

`SGGetOutputNextOutput` (page 2149)
: Returns the next sequence grabber output for the specified output.

`SGNewOutput` (page 2171)
: Creates a new sequence grabber output.

`SGSetChannelOutput` (page 2191)
: Assigns an output to a channel.

`SGSetOutputFlags` (page 2205)
: Configures an existing sequence grabber output.

`SGSetOutputMaximumOffset` (page 2206)
: Specifies the maximum offset for data written to a specified sequence grabber output.

`SGSetOutputNextOutput` (page 2207)
: Specifies the order in which sequence grabber outputs should be used.

Working With Sequence Grabber Settings

`SGGetChannelSettings` (page 2132)
: Retrieves the current settings of a channel used by the sequence grabber.

`SGGetSettings` (page 2151)
: Retrieves the current settings of all channels used by the sequence grabber.

`SGSetChannelSettings` (page 2193)
: Configures a sequence grabber channel.

`SGSetSettings` (page 2208)
: Configures a sequence grabber and its channels.

`SGSettingsDialog` (page 2214)
: Causes a sequence grabber to display its settings dialog box to the user.

Working With Sound Channels

`SGGetSoundInputDriver` (page 2151)
: Determines the sound input device currently in use by a sound channel component.

`SGGetSoundInputParameters` (page 2152)
: Retrieves various parameters that relate to sound recording.

`SGGetSoundInputRate` (page 2153)
: Determines the rate at which the sound channel is collecting sound data.

`SGGetSoundRecordChunkSize` (page 2153)
: Determines the amount of sound data the sequence grabber component works with at a time.

`SGSetSoundInputDriver` (page 2210)
: Assigns a sound input device to a sound channel.

`SGSetSoundInputParameters` (page 2210)
: Sets various parameters that relate to sound recording.

`SGSetSoundInputRate` (page 2211)
: Sets the rate at which the sound channel obtains its sound data.

`SGSetSoundRecordChunkSize` (page 2211)
: Controls the amount of sound data in each group of sound samples during a record operation.

`SGSoundInputDriverChanged` (page 2221)
: Notifies the sequence grabber component whenever you change the configuration of a sound channel's sound input device.

Working With Video Channels

`SGGetCompressBuffer` (page 2136)
: Returns information about the filter buffer established for a video channel.

`SGGetFrameRate` (page 2141)
: Retrieves a video channel's frame rate for recording.

`SGGetSrcVideoBounds` (page 2154)
: Determines the size of the source video boundary rectangle.

`SGGetUseScreenBuffer` (page 2158)
: Determines whether a video channel is allowed to use an offscreen buffer.

`SGGetVideoCompressor` (page 2159)
: Determines a channel's current image compression parameters.

`SGGetVideoCompressorType` (page 2161)
: Determines the type of image compression that is being applied to a channel's video data.

`SGGetVideoDigitizerComponent` (page 2161)
: Determines the video digitizer component that is providing source video to a video channel component.

`SGGetVideoRect` (page 2162)
: Determines the portion of the source video image that is to be captured.

`SGSetCompressBuffer` (page 2196)
: Allows the sequence grabber component to direct your component to create a filter buffer for your video channel.

`SGSetFrameRate` (page 2202)
: Specifies a video channel's frame rate for recording.

`SGSetUseScreenBuffer` (page 2216)
: Controls whether a video channel uses an offscreen buffer.

`SGSetVideoCompressor` (page 2218)
: Specifies many of the parameters that control image compression of the video data captured by a video channel.

`SGSetVideoCompressorType` (page 2219)
: Specifies the type of image compression to be applied to captured video images.

`SGSetVideoDigitizerComponent` (page 2219)
: Assigns a video digitizer component to a video channel.

`SGSetVideoRect` (page 2220)
: Specifies a part of the source video image that is to be captured by a sequence grabber component.

`SGVideoDigitizerChanged` (page 2225)
: Notifies the sequence grabber component whenever you change the configuration of a video channel's video digitizer.

Supporting Functions

`DisposeSGAddFrameBottleUPP` (page 2096)
: Disposes of an SGAddFrameBottleUPP pointer.

`DisposeSGCompressBottleUPP` (page 2096)
: Disposes of an SGCompressBottleUPP pointer.

`DisposeSGCompressCompleteBottleUPP` (page 2097)
: Disposes of an SGCompressCompleteBottleUPP pointer.

`DisposeSGDataUPP` (page 2097)
: Disposes of an SGDataUPP pointer.

`DisposeSGDisplayBottleUPP` (page 2098)
: Disposes of an SGDisplayBottleUPP pointer.

`DisposeSGDisplayCompressBottleUPP` (page 2098)
: Disposes of an SGDisplayCompressBottleUPP pointer.

`DisposeSGGrabBottleUPP` (page 2099)
: Disposes of an SGGrabBottleUPP pointer.

`DisposeSGGrabCompleteBottleUPP` (page 2099)
: Disposes of an SGGrabCompleteBottleUPP pointer.

`DisposeSGGrabCompressCompleteBottleUPP` (page 2099)
: Disposes of an SGGrabCompressCompleteBottleUPP pointer.

`DisposeSGModalFilterUPP` (page 2100)
: Disposes of an SGModalFilterUPP pointer.

`DisposeSGTransferFrameBottleUPP` (page 2100)
: Disposes of an SGTransferFrameBottleUPP pointer.

`NewSGAddFrameBottleUPP` (page 2101)
: Allocates a Universal Procedure Pointer for the SGAddFrameBottleProc callback.

`NewSGCompressBottleUPP` (page 2101)
: Allocates a Universal Procedure Pointer for the SGCompressBottleProc callback.

`NewSGCompressCompleteBottleUPP` (page 2102)
: Allocates a Universal Procedure Pointer for the SGCompressCompleteBottleProc callback.

`NewSGDataUPP` (page 2102)
: Allocates a Universal Procedure Pointer for the SGDataProc callback.

`NewSGDisplayBottleUPP` (page 2103)
: Allocates a Universal Procedure Pointer for the SGDisplayBottleProc callback.

`NewSGDisplayCompressBottleUPP` (page 2104)
: Allocates a Universal Procedure Pointer for the SGDisplayCompressBottleProc callback.

`NewSGGrabBottleUPP` (page 2104)
: Allocates a Universal Procedure Pointer for the SGGrabBottleProc callback.

`NewSGGrabCompleteBottleUPP` (page 2105)
: Allocates a Universal Procedure Pointer for the SGGrabCompleteBottleProc callback.

`NewSGGrabCompressCompleteBottleUPP` (page 2105)
: Allocates a Universal Procedure Pointer for the SGGrabCompressCompleteBottleProc callback.

`NewSGModalFilterUPP` (page 2106)
: Allocates a Universal Procedure Pointer for the SGModalFilterProc callback.

`NewSGTransferFrameBottleUPP` (page 2106)
: Allocates a Universal Procedure Pointer for the SGTransferFrameBottleProc callback.

`SGAddFrame` (page 2109)
: Provides default values for your add-frame function.

`SGChannelGetCodecSettings` (page 2114)
: Gets the codec settings for a sequence grabber channel.

`SGChannelPutPicture` (page 2116)
: Undocumented

`SGChannelSetCodecSettings` (page 2116)
: Sets the codec settings for a sequence grabber channel.

`SGCompressFrame` (page 2118)
: Provides the default behavior for your compress function.

`SGCompressFrameComplete` (page 2119)
: Provides the default behavior for your compress-complete function.

`SGDisplayCompress` (page 2120)
: Provides the default behavior for your display-compress function.

`SGDisplayFrame` (page 2121)
: Provides the default behavior for your display function.

`SGGetBufferInfo` (page 2125)
: Obtains information about a buffer that has been passed to a callback function.

`SGGetChannelTimeBase` (page 2133)
: Retrieves a reference to the time base that is being used by a sequence grabber channel.

`SGGetDataOutputStorageSpaceRemaining64` (page 2138)
: Provides a 64-bit version of SGGetDataOutputStorageSpaceRemaining.

`SGGetInstrument` (page 2143)
: Gets a tone description for a music sequence grabber channel.

`SGGetStorageSpaceRemaining64` (page 2155)
: Provides a 64-bit version of SGGetStorageSpaceRemaining.

`SGGrabCompressComplete` (page 2163)
: Provides the default behavior for your grab-compress-complete function.

`SGGrabFrame` (page 2163)
: Provides the default behavior for your grab function.

`SGGrabFrameComplete` (page 2164)
: Provides the default behavior for your grab-complete function.

`SGHandleUpdateEvent` (page 2166)
: Requests that a sequence grabber handle an update event.

`SGSetChannelDeviceInput` (page 2189)
: Undocumented

`SGSetInstrument` (page 2204)
Sets a tone description for a music sequence grabber channel.

`SGTransferFrameForCompress` (page 2223)
Provides the default behavior for your transfer-frame function.

Functions

DisposeSGAddFrameBottleUPP

Disposes of an SGAddFrameBottleUPP pointer.

```
void DisposeSGAddFrameBottleUPP (
   SGAddFrameBottleUPP userUPP
);
```

Parameters

userUPP

An `SGAddFrameBottleUPP` pointer. See `Universal Procedure Pointers`.

Return Value

You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes

Introduced in QuickTime 4.1.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

DisposeSGCompressBottleUPP

Disposes of an SGCompressBottleUPP pointer.

```
void DisposeSGCompressBottleUPP (
   SGCompressBottleUPP userUPP
);
```

Parameters

userUPP

An `SGCompressBottleUPP` pointer. See `Universal Procedure Pointers`.

Return Value

You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes

Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

DisposeSGCompressCompleteBottleUPP

Disposes of an SGCompressCompleteBottleUPP pointer.

```
void DisposeSGCompressCompleteBottleUPP (
   SGCompressCompleteBottleUPP userUPP
);
```

Parameters

`userUPP`
An `SGCompressCompleteBottleUPP` pointer. See `Universal Procedure Pointers`.

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

DisposeSGDataUPP

Disposes of an SGDataUPP pointer.

```
void DisposeSGDataUPP (
   SGDataUPP userUPP
);
```

Parameters

`userUPP`
An `SGDataUPP` pointer. See `Universal Procedure Pointers`.

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
MungSaver

VideoProcessing

Declared In
`QuickTimeComponents.h`

DisposeSGDisplayBottleUPP

Disposes of an SGDisplayBottleUPP pointer.

```
void DisposeSGDisplayBottleUPP (
   SGDisplayBottleUPP userUPP
);
```

Parameters

`userUPP`

An `SGDisplayBottleUPP` pointer. See `Universal Procedure Pointers`.

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

DisposeSGDisplayCompressBottleUPP

Disposes of an SGDisplayCompressBottleUPP pointer.

```
void DisposeSGDisplayCompressBottleUPP (
   SGDisplayCompressBottleUPP userUPP
);
```

Parameters

`userUPP`

An `SGDisplayCompressBottleUPP` pointer. See `Universal Procedure Pointers`.

Return Value
You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes
Introduced in QuickTime 4.1.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

DisposeSGGrabBottleUPP

Disposes of an SGGrabBottleUPP pointer.

```
void DisposeSGGrabBottleUPP (
   SGGrabBottleUPP userUPP
);
```

Parameters

userUPP

An `SGGrabBottleUPP` pointer. See `Universal Procedure Pointers`.

Return Value

You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes

Introduced in QuickTime 4.1.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

DisposeSGGrabCompleteBottleUPP

Disposes of an SGGrabCompleteBottleUPP pointer.

```
void DisposeSGGrabCompleteBottleUPP (
   SGGrabCompleteBottleUPP userUPP
);
```

Parameters

userUPP

An `SGGrabCompleteBottleUPP` pointer. See `Universal Procedure Pointers`.

Return Value

You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes

Introduced in QuickTime 4.1.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

DisposeSGGrabCompressCompleteBottleUPP

Disposes of an SGGrabCompressCompleteBottleUPP pointer.

```
void DisposeSGGrabCompressCompleteBottleUPP (
   SGGrabCompressCompleteBottleUPP userUPP
);
```

Parameters

userUPP

An `SGGrabCompressCompleteBottleUPP` pointer. See `Universal Procedure Pointers`.

Return Value

You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes

Introduced in QuickTime 4.1.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

DisposeSGModalFilterUPP

Disposes of an SGModalFilterUPP pointer.

```
void DisposeSGModalFilterUPP (
   SGModalFilterUPP userUPP
);
```

Parameters

userUPP

An `SGModalFilterUPP` pointer. See `Universal Procedure Pointers`.

Return Value

You can access this function's error returns through `GetMoviesError` (page 221) and `GetMoviesStickyError` (page 222).

Version Notes

Introduced in QuickTime 4.1.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qtcapture

qtcapture.win

Declared In

`QuickTimeComponents.h`

DisposeSGTransferFrameBottleUPP

Disposes of an SGTransferFrameBottleUPP pointer.

```
void DisposeSGTransferFrameBottleUPP (
   SGTransferFrameBottleUPP userUPP
);
```

Parameters

userUPP

An SGTransferFrameBottleUPP pointer. See Universal Procedure Pointers.

Return Value

You can access this function's error returns through GetMoviesError (page 221) and GetMoviesStickyError (page 222).

Version Notes

Introduced in QuickTime 4.1.

Availability

Available in Mac OS X v10.0 and later.

Declared In

QuickTimeComponents.h

NewSGAddFrameBottleUPP

Allocates a Universal Procedure Pointer for the SGAddFrameBottleProc callback.

```
SGAddFrameBottleUPP NewSGAddFrameBottleUPP (
   SGAddFrameBottleProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your application-defined function.

Return Value

A new UPP; see Universal Procedure Pointers.

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes

Introduced in QuickTime 4.1. Replaces NewSGAddFrameBottleProc.

Availability

Available in Mac OS X v10.0 and later.

Declared In

QuickTimeComponents.h

NewSGCompressBottleUPP

Allocates a Universal Procedure Pointer for the SGCompressBottleProc callback.

```
SGCompressBottleUPP NewSGCompressBottleUPP (
   SGCompressBottleProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your application-defined function.

Return Value

A new UPP; see `Universal Procedure Pointers`.

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes

Introduced in QuickTime 4.1. Replaces `NewSGCompressBottleProc`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

NewSGCompressCompleteBottleUPP

Allocates a Universal Procedure Pointer for the SGCompressCompleteBottleProc callback.

```
SGCompressCompleteBottleUPP NewSGCompressCompleteBottleUPP (
   SGCompressCompleteBottleProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your application-defined function.

Return Value

A new UPP; see `Universal Procedure Pointers`.

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes

Introduced in QuickTime 4.1. Replaces `NewSGCompressCompleteBottleProc`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

NewSGDataUPP

Allocates a Universal Procedure Pointer for the SGDataProc callback.

```
SGDataUPP NewSGDataUPP (
   SGDataProcPtr userRoutine
);
```

Parameters

userRoutine
A pointer to your application-defined function.

Return Value

A new UPP; see `Universal Procedure Pointers`.

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes

Introduced in QuickTime 4.1. Replaces `NewSGDataProc`.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

BrideOfMungGrab

CaptureAndCompressIPBMovie

Cocoa - SGDataProc

SGDataProcSample

VideoProcessing

Declared In

`QuickTimeComponents.h`

NewSGDisplayBottleUPP

Allocates a Universal Procedure Pointer for the SGDisplayBottleProc callback.

```
SGDisplayBottleUPP NewSGDisplayBottleUPP (
   SGDisplayBottleProcPtr userRoutine
);
```

Parameters

userRoutine
A pointer to your application-defined function.

Return Value

A new UPP; see `Universal Procedure Pointers`.

Discussion

This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes

Introduced in QuickTime 4.1. Replaces `NewSGDisplayBottleProc`.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

NewSGDisplayCompressBottleUPP

Allocates a Universal Procedure Pointer for the SGDisplayCompressBottleProc callback.

```
SGDisplayCompressBottleUPP NewSGDisplayCompressBottleUPP (
   SGDisplayCompressBottleProcPtr userRoutine
);
```

Parameters

`userRoutine`
    A pointer to your application-defined function.

Return Value
A new UPP; see `Universal Procedure Pointers`.

Discussion
This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes
Introduced in QuickTime 4.1. Replaces `NewSGDisplayCompressBottleProc`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

NewSGGrabBottleUPP

Allocates a Universal Procedure Pointer for the SGGrabBottleProc callback.

```
SGGrabBottleUPP NewSGGrabBottleUPP (
   SGGrabBottleProcPtr userRoutine
);
```

Parameters

`userRoutine`
    A pointer to your application-defined function.

Return Value
A new UPP; see `Universal Procedure Pointers`.

Discussion
This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes
Introduced in QuickTime 4.1. Replaces `NewSGGrabBottleProc`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

NewSGGrabCompleteBottleUPP

Allocates a Universal Procedure Pointer for the SGGrabCompleteBottleProc callback.

```
SGGrabCompleteBottleUPP NewSGGrabCompleteBottleUPP (
   SGGrabCompleteBottleProcPtr userRoutine
);
```

Parameters

`userRoutine`

A pointer to your application-defined function.

Return Value
A new UPP; see `Universal Procedure Pointers`.

Discussion
This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes
Introduced in QuickTime 4.1. Replaces `NewSGGrabCompleteBottleProc`.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Sequence Grabbing

Declared In
`QuickTimeComponents.h`

NewSGGrabCompressCompleteBottleUPP

Allocates a Universal Procedure Pointer for the SGGrabCompressCompleteBottleProc callback.

```
SGGrabCompressCompleteBottleUPP NewSGGrabCompressCompleteBottleUPP (
   SGGrabCompressCompleteBottleProcPtr userRoutine
);
```

Parameters

`userRoutine`

A pointer to your application-defined function.

Return Value
A new UPP; see `Universal Procedure Pointers`.

Discussion
This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes
Introduced in QuickTime 4.1. Replaces `NewSGGrabCompressCompleteBottleProc`.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
BrideOfMungGrab

Declared In
`QuickTimeComponents.h`

NewSGModalFilterUPP

Allocates a Universal Procedure Pointer for the SGModalFilterProc callback.

```
SGModalFilterUPP NewSGModalFilterUPP (
   SGModalFilterProcPtr userRoutine
);
```

Parameters

`userRoutine`
A pointer to your application-defined function.

Return Value
A new UPP; see `Universal Procedure Pointers`.

Discussion
This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes
Introduced in QuickTime 4.1. Replaces `NewSGModalFilterProc`.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
qtcapture
qtcapture.win

Declared In
`QuickTimeComponents.h`

NewSGTransferFrameBottleUPP

Allocates a Universal Procedure Pointer for the SGTransferFrameBottleProc callback.

```
SGTransferFrameBottleUPP NewSGTransferFrameBottleUPP (
   SGTransferFrameBottleProcPtr userRoutine
);
```

Parameters

`userRoutine`
A pointer to your application-defined function.

Return Value
A new UPP; see `Universal Procedure Pointers`.

Discussion
This function is used with Macintosh PowerPC systems. See *Inside Macintosh: PowerPC System Software*.

Version Notes
Introduced in QuickTime 4.1. Replaces `NewSGTransferFrameBottleProc`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGAddExtendedFrameReference

Stores extended sample references for a channel component.

```
ComponentResult SGAddExtendedFrameReference (
   SeqGrabComponent s,
   SeqGrabExtendedFrameInfoPtr frameInfo
);
```

Parameters

s

An instance of the sequence grabber component connected to your channel component. The sequence grabber component provides this value through `SGInitChannel` (page 2167).

frameInfo

A pointer to a `SeqGrabExtendedFrameInfo` structure. Your component must place the appropriate information into this structure.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function differs from `SGAddFrameReference` (page 2110) in that it uses a `SeqGrabExtendedFrameInfo` structure instead of a `SeqGrabFrameInfo` structure.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGAddExtendedMovieData

Adds data to a movie without writing data to a movie file.

```
ComponentResult SGAddExtendedMovieData (
   SeqGrabComponent s,
   SGChannel c,
   Ptr p,
   long len,
   wide *offset,
   long chRefCon,
   TimeValue time,
   short writeType,
   SGOutput *whichOutput
);
```

Parameters

s

An instance of the sequence grabber component connected to your channel component. The sequence grabber component provides this value through `SGInitChannel` (page 2167).

c

Identifies the connection to your channel.

p

The location of the data to be added to the movie.

len

The number of bytes of data to be added to the movie.

offset

A pointer to a wide integer that receives the offset to the new data in the movie. If the movie is in memory, the returned offset reflects the location the data will have in the movie on a permanent storage device.

chRefCon

The reference constant for your channel.

time

The time at which the frame was captured, expressed in the time scale associated with your channel.

writeType

A constant (see below) that determines the type of write operation to be used. See these constants:

`seqGrabWriteAppend`

`seqGrabWriteReserve`

`seqGrabWriteFill`

whichOutput

The use of `whichOutput` depends on the value passed in the `writeType` parameter. If `writeType` is `seqGrabWriteAppend` or `seqGrabWriteReserve`, the `whichOutput` parameter is a return value specifying the sequence grabber output to which data was written or in which space was reserved. If `writeType` is `seqGrabWriteFill`, the `whichOutput` parameter is an input value indicating which sequence grabber output the data should be written to.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function differs from `SGAddMovieData` (page 2110) in two respects: the `offset` parameter allows a 64-bit value, and the `whichOutput` parameter does not exist in `SGAddMovieData`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGAddFrame

Provides default values for your add-frame function.

```
ComponentResult SGAddFrame (
   SGChannel c,
   short bufferNum,
   TimeValue atTime,
   TimeScale scale,
   const SGCompressInfo *ci
);
```

Parameters

`c`
The reference that identifies the channel for this operation. The sequence grabber component provides this value to your add-frame function.

`bufferNum`
Identifies the buffer. The sequence grabber component provides this value to your add-frame function.

`atTime`
The time at which the frame was captured, in the time scale specified by the `scale` parameter. The sequence grabber component provides this value to your add-frame function. Your add-frame function can change this value before calling this function. You can determine the duration of a frame by subtracting its capture time from the capture time of the next frame in the sequence.

`scale`
The time scale of the movie. The sequence grabber component provides this value to your add-frame function.

`ci`
A pointer to a `SGCompressInfo` structure. This structure contains information describing the compression characteristics of the image to be added to the movie.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
You should call this function only from your add-frame function. If you call it at any other time, results are unpredictable.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGAddFrameReference

Stores sample references for a channel component.

```
ComponentResult SGAddFrameReference (
   SeqGrabComponent s,
   SeqGrabFrameInfoPtr frameInfo
);
```

Parameters

s

An instance of the sequence grabber component connected to your channel component. The sequence grabber component provides this value through `SGInitChannel` (page 2167).

frameInfo

A pointer to a `SeqGrabFrameInfo` structure. Your component must completely specify the reference by placing the appropriate information into the record referred to by this parameter.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SGAddMovieData

Lets a channel component add data to a movie.

```
ComponentResult SGAddMovieData (
   SeqGrabComponent s,
   SGChannel c,
   Ptr p,
   long len,
   long *offset,
   long chRefCon,
   TimeValue time,
   short writeType
);
```

Parameters

s

An instance of the sequence grabber component connected to your channel component. The sequence grabber component provides this value through `SGInitChannel` (page 2167).

c

Identifies the connection to your channel.

p

The location of the data to be added to the movie.

len

Indicates the number of bytes of data to be added to the movie.

offset

A pointer to a field that is to receive the offset to the new data in the movie. The sequence grabber component returns an offset that is correct in the context of the movie resource, even if the movie is currently stored in memory. That is, if the movie is in memory, the returned offset reflects the location that the data will have in a movie on a permanent storage device, such as a disk.

chRefCon

Your channel's reference constant.

time

The time at which your channel captured the frame. This time value is expressed in your channel's time scale.

writeType

A constant (see below) that determines the type of write operation to be used. See these constants:

`seqGrabWriteAppend`

`seqGrabWriteReserve`

`seqGrabWriteFill`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function combines the services provided by `SGWriteMovieData` (page 2227) and `SGAddFrameReference` (page 2110). Your channel component should not write data directly to the movie file; use this function instead.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

AlwaysPreview

Declared In

`QuickTimeComponents.h`

SGAddOutputDataRefToMedia

Manages capture sessions that involve multiple data references.

```
ComponentResult SGAddOutputDataRefToMedia (
   SeqGrabComponent s,
   SGOutput sgOut,
   Media theMedia,
   SampleDescriptionHandle desc
);
```

Parameters

s

An instance of the sequence grabber component connected to your channel component. The sequence grabber component provides this value through `SGInitChannel` (page 2167).

sgOut
A pointer to the current sequence grabber output.

theMedia
The media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` (page 1630) and `GetTrackMedia` (page 1612). See `Media Identifiers`.

desc
A handle to a `SampleDescription` structure that contains an index, which is assigned to the data by this function.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function is usually called from the `SGWriteSamples` (page 2227) function of a sequence grabber channel component. You pass to it a sequence grabber output along with a media and `SampleDescription` structure, and it adds the data reference to the data reference list of the specified media. It also updates the data reference index field of the `SampleDescription` structure to refer to the data reference.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGAlignChannelRect

Determines whether or not a channel prefers to draw at a particular screen location.

```
ComponentResult SGAlignChannelRect (
   SGChannel c,
   Rect *r
);
```

Parameters

c
The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

r
A pointer to a `Rect` structure. On entry, this structure contains coordinates at which the sequence grabber would like to draw your captured video image. If your component can draw more efficiently or at a higher frame rate at a different location, update the contents of this structure to reflect where you would prefer to draw. The rectangle will be passed in with global, not local, coordinates.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function is called by a sequence grabber to determine whether or not a channel prefers to draw at a particular screen location.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGAppendDeviceListToMenu

Places a list of device names into a specified menu.

```
ComponentResult SGAppendDeviceListToMenu (
   SeqGrabComponent s,
   SGDeviceList list,
   MenuRef mh
);
```

Parameters

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` or `OpenComponent`.

list

A pointer to a pointer to an `SGDeviceListRecord`. The sequence grabber appends the name of each device in the list to the menu specified by the `mh` parameter. If the `sgDeviceNameFlagDeviceUnavailable` flag is set to 1 for a device in the list, the sequence grabber disables the menu item corresponding to that device.

mh

A handle to the menu to which the device names are to be appended.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGChangedSource

Informs the sequence grabber that a component is now using a different device.

```
ComponentResult SGChangedSource (
   SeqGrabComponent s,
   SGChannel c
);
```

Parameters

s

An instance of the sequence grabber component connected to your channel component. The sequence grabber component provides this value through `SGInitChannel` (page 2167).

c

Identifies the connection to your channel.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SGChannelGetCodecSettings

Gets the codec settings for a sequence grabber channel.

```
ComponentResult SGChannelGetCodecSettings (
   SGChannel c,
   Handle *settings
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

settings

A pointer to a handle that the codec should resize and fill in with the current internal settings. These settings are codec-defined and usually opaque. Don't dispose of this handle.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SGChannelGetDataSourceName

Returns the data source name for a track.

```
ComponentResult SGChannelGetDataSourceName (
   SGChannel c,
   Str255 name,
   ScriptCode *scriptTag
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

name

A string that is to receive the source identification information. Set this parameter to `NIL` if you do not want to retrieve the name.

scriptTag

A field that is to receive the source information's language code; see `Localization Codes`. Set this parameter to `NIL` if you do not want this information.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function allows you to get the source information specified with `SGChannelSetDataSourceName` (page 2117).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SGChannelGetRequestedDataRate

Returns the current maximum data rate requested for a channel.

```
ComponentResult SGChannelGetRequestedDataRate (
   SGChannel c,
   long *bytesPerSecond
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

bytesPerSecond

Points to a field that is to receive the maximum data rate requested by the sequence grabber component. This field is set to 0 if the sequence grabber has not set any restrictions.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function allows the sequence grabber component to retrieve the current maximum data rate value from your channel component.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGChannelPutPicture

Undocumented

```
ComponentResult SGChannelPutPicture (
   SGChannel c
);
```

Parameters

c
: The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGChannelSetCodecSettings

Sets the codec settings for a sequence grabber channel.

```
ComponentResult SGChannelSetCodecSettings (
   SGChannel c,
   Handle settings
);
```

Parameters

c
: The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

settings

Undocumented

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGChannelSetDataSourceName

Sets the data source name for a track.

```
ComponentResult SGChannelSetDataSourceName (
   SGChannel c,
   ConstStr255Param name,
   ScriptCode scriptTag
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

name

A string that contains the source identification information. The source information identifies the source of the video data (for example, a videotape name).

scriptTag

The language of the source identification information; see `Localization Codes`.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function allows you to set the source information for a sequence grabber channel. You must set this information before you start digitizing. The sequence grabber channel stores this information in a timecode track in the movie created after the capture is complete. If the video digitizer does not provide timecode information, the sequence grabber does not save this information.

Special Considerations

This function is currently supported only by video channels.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGChannelSetRequestedDataRate

Specifies the maximum requested data rate for a channel.

```
ComponentResult SGChannelSetRequestedDataRate (
   SGChannel c,
   long bytesPerSecond
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

bytesPerSecond

The maximum data rate requested by the sequence grabber component, in bytes per second. The sequence grabber component sets this parameter to 0 to remove any data-rate restrictions.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function allows the sequence grabber component to specify the maximum rate at which it would like to receive data from your channel component. The data rate supplied by the sequence grabber component represents a requested data rate; your component may not be able to observe that rate under all conditions.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SGCompressFrame

Provides the default behavior for your compress function.

```
ComponentResult SGCompressFrame (
   SGChannel c,
   short bufferNum
);
```

Parameters

c

The reference that identifies the channel for this operation. The sequence grabber provides this value to your compress function.

bufferNum

The buffer. The sequence grabber provides this value to your compress function.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Special Considerations

You should call this function only from your compress function. If you call it at any other time, results are unpredictable.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SGCompressFrameComplete

Provides the default behavior for your compress-complete function.

```
ComponentResult SGCompressFrameComplete (
   SGChannel c,
   short bufferNum,
   Boolean *done,
   SGCompressInfo *ci
);
```

Parameters

c

The reference that identifies the channel for this operation. The sequence grabber component provides this value to your compress-complete function.

bufferNum

Identifies the buffer. The sequence grabber component provides this value to your compress-complete function.

done

A pointer to a Boolean value. The function sets this Boolean value to TRUE if the compression is complete, or FALSE if the operation is incomplete. The sequence grabber component provides this pointer to your compress-complete function.

ci

A pointer to a `SGCompressInfo` structure. If the compression is complete, the function completely formats this structure with information that is appropriate to the frame just compressed. The sequence grabber component provides this pointer to your compress-complete function.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Special Considerations

You should call this function only from your compress-complete function. If you call it at any other time, results are unpredictable.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGDisplayCompress

Provides the default behavior for your display-compress function.

```
ComponentResult SGDisplayCompress (
   SGChannel c,
   Ptr dataPtr,
   ImageDescriptionHandle desc,
   MatrixRecord *mp,
   RgnHandle clipRgn
);
```

Parameters

c
: Identifies the channel for this operation. The sequence grabber provides this value to your display-compress function.

dataPtr
: A pointer to the compressed image data. The sequence grabber provides this pointer to your display-compress function.

desc
: A handle to the `ImageDescription` structure to use for the decompression operation. The sequence grabber provides this handle to your display-compress function.

mp
: A pointer to a `MatrixRecord` structure. This structure contains the transformation matrix to use when displaying the image. If there is no matrix for the operation, set this parameter to `NIL`.

clipRgn
: A handle to a `MacRegion` structure that defines the clipping region for the destination image. This region is defined in the destination coordinate system. If there is no clipping region, set this parameter to `NIL`.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Special Considerations

You should call this function only from your display-compress function. If you call it at any other time, results are unpredictable.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGDisplayFrame

Provides the default behavior for your display function.

```
ComponentResult SGDisplayFrame (
   SGChannel c,
   short bufferNum,
   const MatrixRecord *mp,
   RgnHandle clipRgn
);
```

Parameters

c

The reference that identifies the channel for this operation. The sequence grabber component provides this value to your display function.

bufferNum

Identifies the buffer. The sequence grabber component provides this value to your display function.

mp

A pointer to a `MatrixRecord` structure for the display operation. If there is no matrix for the operation, set this parameter to `NIL`.

clipRgn

A handle to a `MacRegion` structure that defines the clipping region for the destination image. This region is defined in the destination coordinate system. If there is no clipping region, set this parameter to `NIL`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Special Considerations

You should call this function only from your display function. If you call it at any other time, results are unpredictable.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SGDisposeChannel

Removes a channel from a sequence grabber component.

```
ComponentResult SGDisposeChannel (
   SeqGrabComponent s,
   SGChannel c
);
```

Parameters

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` or `OpenComponent`.

c
: The reference that identifies the channel you want to close. You obtain this reference from `SGNewChannel` (page 2169).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
MovieGWorlds

qtcapture

Sequence Grabbing

SGDataProcSample

WhackedTV

Declared In
`QuickTimeComponents.h`

SGDisposeDeviceList

Disposes of a device list.

```
ComponentResult SGDisposeDeviceList (
   SeqGrabComponent s,
   SGDeviceList list
);
```

Parameters

s
: The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` or `OpenComponent`.

list
: A pointer to a pointer to an `SGDeviceListRecord` structure. The sequence grabber disposes of the memory used by this structure.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
SGDevices

WhackedTV

Declared In

`QuickTimeComponents.h`

SGDisposeOutput

Disposes of an existing sequence grabber output.

```
ComponentResult SGDisposeOutput (
   SeqGrabComponent s,
   SGOutput sgOut
);
```

Parameters

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` or `OpenComponent`.

sgOut

Identifies the sequence grabber output for this operation. You obtain this identifier by calling `SGNewOutput` (page 2171).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Use this function to dispose of an existing output. If any sequence grabber channels are using this output, the sequence grabber component assigns them to an undefined output.

Special Considerations

You cannot dispose of an output when the sequence grabber component is in record mode.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SGGetAdditionalSoundRates

Returns the additional sound sample rates added to a specified sequence grabber sound channel.

```
ComponentResult SGGetAdditionalSoundRates (
   SGChannel c,
   Handle *rates
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

rates

A pointer to the handle where the list of additional sample rates should be returned. If no additional sample rates have been set, this function sets the rates handle to `NIL`. The caller of this routine is responsible for disposing of the returned handle.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SGGetAlignmentProc

Obtains information about the best screen positions for a sequence grabber's video image in terms of appearance and maximum performance.

```
ComponentResult SGGetAlignmentProc (
   SeqGrabComponent s,
   ICMAlignmentProcRecordPtr alignmentProc
);
```

Parameters

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` or `OpenComponent`.

alignmentProc

A pointer to an `ICMAlignmentProcRecord` structure. The sequence grabber places its alignment information into this structure.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

BrideOfMungGrab

DigitizerShell

hacktv

hacktv.win

SGDataProcSample

Declared In

`QuickTimeComponents.h`

SGGetBufferInfo

Obtains information about a buffer that has been passed to a callback function.

```
ComponentResult SGGetBufferInfo (
   SGChannel c,
   short bufferNum,
   PixMapHandle *bufferPM,
   Rect *bufferRect,
   GWorldPtr *compressBuffer,
   Rect *compressBufferRect
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

bufferNum

Identifies the buffer. The sequence grabber component provides this value to your callback function.

bufferPM

A pointer to a location that is to receive a handle to the `PixMap` structure that contains the image. Note that this structure may be offscreen. Do not dispose of this structure. If you do not want this information, set this parameter to `NIL`.

bufferRect

A pointer to a `Rect` structure that is to receive the dimensions of the image's boundary rectangle. If you do not want this information, set this parameter to `NIL`.

compressBuffer

A pointer to a location that is to receive a pointer to the filter buffer for the image. The sequence grabber component returns this information only if your application has assigned a filter buffer to this video channel. You assign a filter buffer by calling `SGSetCompressBuffer` (page 2196). Do not dispose of this buffer.

compressBufferRect

A pointer to a `Rect` structure that is to receive the dimensions of the filter buffer for the image. The sequence grabber component returns this information only if your application has assigned a filter buffer to this video channel. You assign a filter buffer by calling `SGSetCompressBuffer` (page 2196). If you have not assigned a filter buffer, the sequence grabber component returns an empty rectangle. If you do not want this information, set this parameter to `NIL`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

DigitizerShell

Sequence Grabbing

Declared In

`QuickTimeComponents.h`

SGGetChannelBounds

Determines a channel's display boundary rectangle.

```
ComponentResult SGGetChannelBounds (
   SGChannel c,
   Rect *bounds
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

bounds

A pointer to a `Rect` structure that is to receive information about your channel's display boundary rectangle.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

hacktv

hacktv.win

WhackedTV

Declared In

`QuickTimeComponents.h`

SGGetChannelClip

Retrieves a channel's clipping region.

```
ComponentResult SGGetChannelClip (
   SGChannel c,
   RgnHandle *theClip
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

theClip

A pointer to a handle that is to receive a `MacRegion` structure that defines the clipping region. The application is responsible for disposing of this handle. If there is no clipping region, set this handle to `NIL`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Special Considerations

Note that some devices may not support clipping.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

AlwaysPreview

Declared In

`QuickTimeComponents.h`

SGGetChannelDeviceAndInputNames

Returns the sequence grabber's current device and input names.

```
ComponentResult SGGetChannelDeviceAndInputNames (
   SGChannel c,
   Str255 outDeviceName,
   Str255 outInputName,
   short *outInputNumber
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

outDeviceName

The current device names for display to the user.

outInputName

The current input names for display to the user.

outInputNumber

A pointer to the number of currently selected inputs.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This is a utility call that lets you find out the sequence grabber's current device and input names, instead of having to call `GetDeviceList` and walk it yourself. Pass `NIL` for parameters you are not interested in.

Version Notes

Introduced in QuickTime 6.

Availability

Available in Mac OS X v10.2 and later.

Related Sample Code

WhackedTV

Declared In
`QuickTimeComponents.h`

SGGetChannelDeviceList

Retrieves a list of the devices that are valid for a specified channel.

```
ComponentResult SGGetChannelDeviceList (
   SGChannel c,
   long selectionFlags,
   SGDeviceList *list
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

selectionFlags

Flags (see below) that control the data you are to return for each device. See these constants:

`sgDeviceListWithIcons`

`sgDeviceListDontCheckAvailability`

`sgDeviceListIncludeInputs`

list

A pointer to a pointer to an `SGDeviceListRecord` structure. The channel creates this structure and returns a pointer to it in the field referred to by this parameter. Applications use `SGDisposeDeviceList` (page 2122) to dispose of the memory used by the list.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function can be useful for retrieving the name of the current device. Retrieve the device list and use the `selectedIndex` field to determine which device is currently in use.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
SGDevices

WhackedTV

Declared In
`QuickTimeComponents.h`

SGGetChannelInfo

Determines how a channel's data is represented to the user: as visual data or audio data, or both.

```
ComponentResult SGGetChannelInfo (
   SGChannel c,
   long *channelInfo
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

channelInfo

A pointer to a long integer that is to receive channel information flags (see below). You may set more than one flag to 1. Set unused flags to 0. See these constants:

`seqGrabHasBounds`

`seqGrabHasVolume`

`seqGrabHasDiscreteSamples`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SGGetChannelMatrix

Retrieves a channel's display transformation matrix.

```
ComponentResult SGGetChannelMatrix (
   SGChannel c,
   MatrixRecord *m
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

m

A pointer to a `MatrixRecord` structure. Place your current matrix values into this structure.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code
AlwaysPreview

Declared In
`QuickTimeComponents.h`

SGGetChannelMaxFrames

Determines the number of frames left to be captured from a specified channel.

```
ComponentResult SGGetChannelMaxFrames (
   SGChannel c,
   long *frameCount
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

frameCount

A pointer to a long integer that is to receive a value specifying the number of frames left to be captured during the preview or record operation. If the returned value is -1, the sequence grabber channel component captures as many frames as it can.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGGetChannelPlayFlags

Retrieves the playback control flags that you set with SGSetChannelPlayFlags.

```
ComponentResult SGGetChannelPlayFlags (
   SGChannel c,
   long *playFlags
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

playFlags

A pointer to a long integer that is to receive flags (see below) that influence channel playback. Set unused flags to 0. See these constants:

`channelPlayNormal`

`channelPlayFast`

`channelPlayHighQuality`

`channelPlayAllData`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

WhackedTV

Declared In

`QuickTimeComponents.h`

SGGetChannelRefCon

Returns a reference constant that was previously set by SGSetChannelRefCon.

```
ComponentResult SGGetChannelRefCon (
   SGChannel c,
   long *refCon
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

refCon

A pointer to the reference constant set by `SGSetChannelRefCon` (page 2192), normally used to point to a data structure containing information your sequence grabber channel needs.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 6.

Availability

Available in Mac OS X v10.2 and later.

Related Sample Code

WhackedTV

Declared In

`QuickTimeComponents.h`

SGGetChannelSampleDescription

Retrieves a channel's sample description structure.

```
ComponentResult SGGetChannelSampleDescription (
   SGChannel c,
   Handle sampleDesc
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

sampleDesc

A handle that is to receive the structure.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The channel returns a structure that is appropriate to the type of data being captured. For video channels, the channel component returns an `ImageDescription` structure; for sound channels, it receives a `SoundDescription` structure.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

AlwaysPreview

BrideOfMungGrab

Cocoa - SGDataProc

SGDataProcSample

WhackedTV

Declared In

`QuickTimeComponents.h`

SGGetChannelSettings

Retrieves the current settings of a channel used by the sequence grabber.

```
ComponentResult SGGetChannelSettings (
   SeqGrabComponent s,
   SGChannel c,
   UserData *ud,
   long flags
);
```

Parameters

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` or `OpenComponent`.

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

ud

On return, a pointer to a `UserDataRecord` structure that contains the configuration information.

flags

Reserved for Apple. Set this parameter to 0.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

MungSaver

WhackedTV

Declared In

`QuickTimeComponents.h`

SGGetChannelTimeBase

Retrieves a reference to the time base that is being used by a sequence grabber channel.

```
ComponentResult SGGetChannelTimeBase (
   SGChannel c,
   TimeBase *tb
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

tb

A pointer to a time base identifier, such as that returned by `NewTimeBase` (page 261).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
BrideOfMungGrab

Declared In
`QuickTimeComponents.h`

SGGetChannelTimeScale

Lets the sequence grabber retrieve a channel's time scale.

```
ComponentResult SGGetChannelTimeScale (
   SGChannel c,
   TimeScale *scale
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

scale

A pointer to a time scale. Your channel component places information about its time scale into this structure.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
The time scale you return typically corresponds to the time scale of the media that has been created by your channel. Applications may use this time scale in their data functions.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
BrideOfMungGrab

CaptureAndCompressIPBMovie

Cocoa - SGDataProc

SGDataProcSample

WhackedTV

Declared In
`QuickTimeComponents.h`

SGGetChannelUsage

Determines how the sequence grabber component is using a channel.

```
ComponentResult SGGetChannelUsage (
   SGChannel c,
   long *usage
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

usage

A pointer to a location that is to receive flags (see below) that specify how your channel is to be used. You may set more than one of these flags to 1. Set unused flags to 0. See these constants:

`seqGrabRecord`

`seqGrabPreview`

`seqGrabPlayDuringRecord`

`seqGrabLowLatencyCapture`

`seqGrabAlwaysUseTimeBase`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier. Flags added in QuickTime 6.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

AlwaysPreview

WhackedTV

Declared In

`QuickTimeComponents.h`

SGGetChannelVolume

Determines a channel's sound volume setting.

```
ComponentResult SGGetChannelVolume (
   SGChannel c,
   short *volume
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

volume

A pointer to an integer that is to receive the volume setting of the channel represented as a 16-bit, fixed-point number. The high-order 8 bits contain the integer part of the value; the low-order 8 bits contain the fractional part. Volume values range from -1.0 to 1.0. Negative values play no sound but preserve the absolute value of the volume setting.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SGGetCompressBuffer

Returns information about the filter buffer established for a video channel.

```
ComponentResult SGGetCompressBuffer (
   SGChannel c,
   short *depth,
   Rect *compressSize
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

depth

A pointer to a field that is to receive the pixel depth of the filter buffer. If your component is not filtering the input video data, set the returned value to 0.

compressSize

A pointer to a `Rect` structure that is to receive the dimensions of the filter buffer. If your component is not filtering the input video data, return an empty rectangle (all coordinates set to 0).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SGGetDataOutput

Determines the movie file that is currently assigned to a sequence grabber component and the control flags that would govern a record operation.

```
ComponentResult SGGetDataOutput (
   SeqGrabComponent s,
   FSSpec *movieFile,
   long *whereFlags
);
```

Parameters

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` or `OpenComponent`.

movieFile

A pointer to an `FSSpec` structure that is to receive information about the movie file for this record operation.

whereFlags

A pointer to a long integer that is to receive flags (see below) that control the record operation. See these constants:

`seqGrabToDisk`

`seqGrabToMemory`

`seqGrabDontUseTempMemory`

`seqGrabAppendToFile`

`seqGrabDontAddMovieResource`

`seqGrabDontMakeMovie`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

You set the characteristics returned by this function by calling `SGSetDataOutput` (page 2197). If you have not set these characteristics before calling this function, the returned data is meaningless.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SGGetDataOutputStorageSpaceRemaining

Returns the amount of space remaining in the data reference associated with an output.

```
ComponentResult SGGetDataOutputStorageSpaceRemaining (
   SeqGrabComponent s,
   SGOutput sgOut,
   unsigned long *space
);
```

Parameters

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` or `OpenComponent`.

sgOut

Identifies the sequence grabber output for this operation. You obtain this identifier by calling `SGNewOutput` (page 2171).

space

A pointer to an unsigned long integer, where the sequence grabber component returns a value that indicates the number of bytes of space remaining in the data reference associated with the output.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Use this function in place of `SGGetStorageSpaceRemaining` (page 2154) in cases where you are working with more than one output.

Version Notes

A sequence grabber output ties a sequence grabber channel to a specified data reference for the output of captured data. If you are capturing to a single movie file, you can continue to use `SGSetDataOutput` (page 2197) or `SGSetDataRef` (page 2199) to specify the sequence grabber's destination. However, if you want to capture movie data into several different files or data references, you must use sequence grabber outputs to do so. Even if you are using outputs, you must still use `SGSetDataOutput` or `SGSetDataRef` to identify where the sequence grabber should create the movie resource. You are responsible for creating outputs, assigning them to sequence grabber channels, and disposing of them when you are done.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SGGetDataOutputStorageSpaceRemaining64

Provides a 64-bit version of SGGetDataOutputStorageSpaceRemaining.

```
ComponentResult SGGetDataOutputStorageSpaceRemaining64 (
   SeqGrabComponent s,
   SGOutput sgOut,
   wide *space
);
```

Parameters

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` or `OpenComponent`.

sgOut
: Identifies the sequence grabber output for this operation. You obtain this identifier by calling `SGNewOutput` (page 2171).

space
: A pointer to a 64-bit wide integer, where the sequence grabber component returns a value that indicates the number of bytes of space remaining in the data reference associated with the output.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGGetDataRate

Determines for a sequence grabber how much recording time is left.

```
ComponentResult SGGetDataRate (
   SGChannel c,
   long *bytesPerSecond
);
```

Parameters

c
: The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

bytesPerSecond
: A pointer to a long integer that is to receive a value indicating the number of bytes your component is recording per second. Your component calculates this value based on its current operational parameters.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGGetDataRef

Determines the data reference currently assigned to a sequence grabber component and the control flags that would govern a record operation.

```
ComponentResult SGGetDataRef (
   SeqGrabComponent s,
   Handle *dataRef,
   OSType *dataRefType,
   long *whereFlags
);
```

Parameters

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` or `OpenComponent`.

dataRef

A pointer to a handle that is to receive the information that identifies the destination container.

dataRefType

A pointer to a field that is to receive the type of data reference.

whereFlags

A pointer to a long integer that is to receive flags (see below) that control the record operation. See these constants:

`seqGrabToDisk`

`seqGrabToMemory`

`seqGrabDontUseTempMemory`

`seqGrabAppendToFile`

`seqGrabDontAddMovieResource`

`seqGrabDontMakeMovie`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function allows you to determine the data reference that is currently assigned to a sequence grabber component and the control flags that would govern a record operation. You set these characteristics by calling `SGSetDataRef` (page 2199). If you have not set these characteristics before calling this function, the returned data is meaningless.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SGGetFlags

Retrieves a sequence grabber's control flags.

```
ComponentResult SGGetFlags (
   SeqGrabComponent s,
   long *sgFlags
);
```

Parameters

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` or `OpenComponent`.

sgFlags

A pointer to a long integer that is to receive the control flag (see below) for the current operation. See these constants:

`sgFlagControlledGrab`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Sequence Grabbing

Declared In

`QuickTimeComponents.h`

SGGetFrameRate

Retrieves a video channel's frame rate for recording.

```
ComponentResult SGGetFrameRate (
   SGChannel c,
   Fixed *frameRate
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

frameRate

A pointer to a field to receive the current frame rate.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGGetGWorld

Determines the graphics port and device for a sequence grabber component.

```
ComponentResult SGGetGWorld (
   SeqGrabComponent s,
   CGrafPtr *gp,
   GDHandle *gd
);
```

Parameters

s
: The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` or `OpenComponent`.

gp
: A pointer to a location that is to receive a pointer to the destination graphics port. Set this parameter to `NIL` if you are not interested in this information.

gd
: A pointer to a location that is to receive a handle to the destination graphics device. Set this parameter to `NIL` if you are not interested in this information.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGGetIndChannel

Collects information about all of the channel components currently in use by a sequence grabber component.

```
ComponentResult SGGetIndChannel (
   SeqGrabComponent s,
   short index,
   SGChannel *ref,
   OSType *chanType
);
```

Parameters

s
: The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` or `OpenComponent`.

index

Specifies an index value that identifies the channel to be queried. The first channel has an index value of 1.

ref

A pointer to a field to receive a value identifying your connection to the channel. If you do not want to receive this information, set this parameter to `NIL`.

chanType

A pointer to a field to receive the channel's subtype value (see below). This value indicates the `media` type supported by the channel component. See these constants:

`VideoMediaType`

`SoundMediaType`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

WhackedTV

Declared In

`QuickTimeComponents.h`

SGGetInstrument

Gets a tone description for a music sequence grabber channel.

```
ComponentResult SGGetInstrument (
   SGChannel c,
   ToneDescription *td
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

td

Pointer to a `ToneDescription` structure.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGGetLastMovieResID

Retrieves the last resource ID used by the sequence grabber component.

```
ComponentResult SGGetLastMovieResID (
   SeqGrabComponent s,
   short *resID
);
```

Parameters

s
: The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` or `OpenComponent`.

resID
: A pointer to an integer that is to receive the resource ID the sequence grabber assigned to the movie resource it just created.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGGetMaximumRecordTime

Determines the time limit you have set for a record operation.

```
ComponentResult SGGetMaximumRecordTime (
   SeqGrabComponent s,
   unsigned long *ticks
);
```

Parameters

s
: The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` or `OpenComponent`.

ticks
: A pointer to a long integer that is to receive a value indicating the maximum duration for the record operation, in system ticks (sixtieths of a second). A value of 0 indicates that there is no time limit.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGGetMode

Determines whether a sequence grabber component is in preview mode or record mode.

```
ComponentResult SGGetMode (
   SeqGrabComponent s,
   Boolean *previewMode,
   Boolean *recordMode
);
```

Parameters

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` or `OpenComponent`.

previewMode

A pointer to a Boolean. The sequence grabber component sets this field to TRUE if the component is in preview mode.

recordMode

A pointer to a Boolean. The sequence grabber component sets this field to TRUE if the component is in record mode.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGGetMovie

Returns a reference to the movie that contains the data collected during a record operation.

```
Movie SGGetMovie (
   SeqGrabComponent s
);
```

Parameters

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` or `OpenComponent`.

Return Value
A movie identifier, such as that returned from `NewMovie` (page 259).

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGGetNextExtendedFrameReference

Allows a channel component to retrieve the sample references stored previously by SGAddExtendedMovieData or SGAddExtendedFrameReference.

```
ComponentResult SGGetNextExtendedFrameReference (
   SeqGrabComponent s,
   SeqGrabExtendedFrameInfoPtr frameInfo,
   TimeValue *frameDuration,
   long *frameNumber
);
```

Parameters

s
An instance of the sequence grabber component connected to your channel component. The sequence grabber component provides this value through `SGInitChannel` (page 2167).

frameInfo
A pointer to a `SeqGrabExtendedFrameInfo` structure. Your component must place the appropriate information into this structure.

frameDuration
A pointer to a time value. The sequence grabber component calculates the duration of the specified frame and returns that duration in this structure. The sequence grabber component cannot calculate the duration of the last frame in a sequence. For the last frame, the time value is set to -1.

frameNumber
A pointer to a long integer representing the frame number. Frame numbers need not be sequential, and need not start at 0. To retrieve information about the first frame in a movie, set the integer to -1.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Your channel component can process frame references sequentially or randomly. You can specify any relative frame for which you want to retrieve information.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGGetNextFrameReference

Lets a channel component retrieve the sample references that were stored by calling SGAddMovieData or SGAddFrameReference.

```
ComponentResult SGGetNextFrameReference (
   SeqGrabComponent s,
   SeqGrabFrameInfoPtr frameInfo,
   TimeValue *frameDuration,
   long *frameNumber
);
```

Parameters

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` or `OpenComponent`.

frameInfo

A pointer to a `SeqGrabFrameInfo` structure. Your component must identify itself to the sequence grabber component by setting the `frameChannel` field of this structure to the component instance that identifies the current connection to your channel. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170). The sequence grabber component then returns information about the specified frame in the remaining fields of this structure.

frameDuration

A pointer to a time value. The sequence grabber component calculates the duration of the specified frame and returns that duration in the structure referred to by this parameter. The sequence grabber component cannot calculate the duration of the last frame in a sequence. In this case, the sequence grabber component sets the returned time value to -1.

frameNumber

A pointer to a long integer. Your channel component specifies the frame number corresponding to the frame about which you want to retrieve information. Frames are numbered starting at 0. However, frame numbers need not start at 0, and they need not be sequential. Set the integer to -1 to retrieve information about the first frame in a movie.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

AlwaysPreview

Declared In

`QuickTimeComponents.h`

SGGetOutputDataReference

Returns information about the data reference associated with the specified sequence grabber output.

```
ComponentResult SGGetOutputDataReference (
   SeqGrabComponent s,
   SGOutput sgOut,
   Handle *dataRef,
   OSType *dataRefType
);
```

Parameters

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` or `OpenComponent`.

sgOut

Identifies the sequence grabber output for this operation. You obtain this identifier by calling `SGNewOutput` (page 2171).

dataRef

A pointer to the handle in which the data reference is returned. If you do not need the data reference, set this parameter to `NIL`.

dataRefType

A pointer in which the type of the data reference is returned. If you do not need this information, set this parameter to `NIL`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The caller is responsible for disposing of the returned handle.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SGGetOutputMaximumOffset

Returns the maximum offset for data written to the specified sequence grabber output.

```
ComponentResult SGGetOutputMaximumOffset (
   SeqGrabComponent s,
   SGOutput sgOut,
   wide *maxOffset
);
```

Parameters

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` or `OpenComponent`.

sgOut

Identifies the current sequence grabber output. You obtain this identifier by calling `SGNewOutput` (page 2171).

maxOffset

A pointer to the value of the maximum offset for data written to this output. This value is initialized to `(2^32-1)` on systems with a 32-bit file system, and `(2^64-1)` on systems with a 64-bit file system.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SGGetOutputNextOutput

Returns the next sequence grabber output for the specified output.

```
ComponentResult SGGetOutputNextOutput (
   SeqGrabComponent s,
   SGOutput sgOut,
   SGOutput *nextOut
);
```

Parameters

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` or `OpenComponent`.

sgOut

Identifies the current sequence grabber output. You obtain this identifier by calling `SGNewOutput` (page 2171).

nextOut

A pointer to the next output to be used. If there is no next output, this value is `NIL`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SGGetPause

Determines whether the sequence grabber is paused.

```
ComponentResult SGGetPause (
   SeqGrabComponent s,
   Byte *paused
);
```

Parameters

s
The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` or `OpenComponent`.

paused
A pointer to a field that is to receive a constant (see below) that indicates whether the sequence grabber is currently paused. See these constants:
`seqGrabUnpause`
`seqGrabPause`
`seqGrabPauseForMenu`

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
WhackedTV

Declared In
`QuickTimeComponents.h`

SGGetPreferredPacketSize

Returns the preferred packet size for the sequence grabber component.

```
ComponentResult SGGetPreferredPacketSize (
   SGChannel c,
   long *preferredPacketSizeInBytes
);
```

Parameters

c
The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

preferredPacketSizeInBytes
The preferred packet size in bytes.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
`SGGetPreferredPacketSize` was added in QuickTime 2.5 to support video conferencing applications.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGGetSettings

Retrieves the current settings of all channels used by the sequence grabber.

```
ComponentResult SGGetSettings (
   SeqGrabComponent s,
   UserData *ud,
   long flags
);
```

Parameters

`s`
: The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` or `OpenComponent`.

`ud`
: A pointer to a space where the sequence grabber returns a pointer to a `UserDataRecord` structure that contains the configuration information. Your application is responsible for disposing of this structure when it is done with it.

`flags`
: Reserved for Apple. Set this parameter to 0.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
WhackedTV

Declared In
`QuickTimeComponents.h`

SGGetSoundInputDriver

Determines the sound input device currently in use by a sound channel component.

```
long SGGetSoundInputDriver (
   SGChannel c
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

Return Value

A reference to the sound input device. If your sound channel is not using a sound input device, returns `NIL`.

Discussion

You may want to gain access to the sound input device if you want to change the device's configuration.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SGGetSoundInputParameters

Retrieves various parameters that relate to sound recording.

```
ComponentResult SGGetSoundInputParameters (
   SGChannel c,
   short *sampleSize,
   short *numChannels,
   OSType *compressionType
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

sampleSize

A pointer to a field to receive the sample size. Set this field to 8 for 8-bit sound or to 16 for 16-bit sound.

numChannels

A pointer to a field to receive the number of sound channels used by the sound sample. Set this field to 1 for monaural sounds or to 2 for stereo sounds.

compressionType

A pointer to a field to receive the format of the sound data (see below). See these constants:

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGGetSoundInputRate

Determines the rate at which the sound channel is collecting sound data.

```
Fixed SGGetSoundInputRate (
   SGChannel c
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

Return Value
A fixed-point number that indicates the number of samples your sound channel collects per second.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGGetSoundRecordChunkSize

Determines the amount of sound data the sequence grabber component works with at a time.

```
long SGGetSoundRecordChunkSize (
   SGChannel c
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

Return Value
A long integer that specifies the number of seconds of sound data your channel works with at a time.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGGetSrcVideoBounds

Determines the size of the source video boundary rectangle.

```
ComponentResult SGGetSrcVideoBounds (
   SGChannel c,
   Rect *r
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

r

A pointer to a `Rect` structure that is to receive information about your channel's source video boundary rectangle.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

For video channel components that work with video digitizer components, the source video boundary rectangle corresponds to the video digitizer's active source rectangle.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

BrideOfMungGrab

hacktv

hacktv.win

qtcapture

qtcapture.win

Declared In

`QuickTimeComponents.h`

SGGetStorageSpaceRemaining

Monitors the amount of space remaining for use during a record operation.

```
ComponentResult SGGetStorageSpaceRemaining (
   SeqGrabComponent s,
   unsigned long *bytes
);
```

Parameters

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` or `OpenComponent`.

bytes

A pointer to a long integer that is to receive a value indicating the amount of space remaining for the current record operation. If you are recording to memory, this value contains information about the amount of memory remaining. If you are recording to a movie file, this value contains information about the amount of storage space available on the device that holds the file.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

You can call this function only after you have started a record operation.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SGGetStorageSpaceRemaining64

Provides a 64-bit version of SGGetStorageSpaceRemaining.

```
ComponentResult SGGetStorageSpaceRemaining64 (
   SeqGrabComponent s,
   wide *bytes
);
```

Parameters

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` or `OpenComponent`.

bytes

A pointer to a wide integer that is to receive a value indicating the amount of space remaining for the current record operation. If you are recording to memory, this value contains information about the amount of memory remaining. If you are recording to a movie file, this value contains information about the amount of storage space available on the device that holds the file.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 4.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SGGetTextReturnToSpaceValue

Indicates whether the text channel component should replace return characters with spaces.

```
ComponentResult SGGetTextReturnToSpaceValue (
   SGChannel c,
   short *rettospace
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

rettospace

A pointer to a 16-bit integer. On return, this parameter is TRUE if the text channel is replacing return characters with spaces, or FALSE if the text channel is not replacing return characters with spaces.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
When you capture text from a closed-caption television source, the text is composed of four lines of text of up to 32 characters each, each line separated by a return character. Sometimes it is useful to replace the return characters with spaces. You can call this function to determine whether the text channel component is replacing return characters with spaces.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGGetTimeBase

Retrieves a reference to the time base that is being used by a sequence grabber component.

```
ComponentResult SGGetTimeBase (
   SeqGrabComponent s,
   TimeBase *tb
);
```

Parameters

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` or `OpenComponent`.

tb

A pointer to a time base identifier, such as that returned by `NewTimeBase` (page 261).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
AlwaysPreview

BrideOfMungGrab

Declared In
`QuickTimeComponents.h`

SGGetTimeRemaining

Obtains an estimate of the amount of recording time that remains for the current record operation.

```
ComponentResult SGGetTimeRemaining (
   SeqGrabComponent s,
   long *ticksLeft
);
```

Parameters

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` or `OpenComponent`.

ticksLeft

A pointer to a long integer that is to receive a value indicating an estimate of the amount of time remaining for the current record operation. This value is expressed in system ticks (sixtieths of a second).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGGetUserVideoCompressorList

Returns the video compression formats to be displayed by the specified sequence grabber video channel.

```
ComponentResult SGGetUserVideoCompressorList (
   SGChannel c,
   Handle *compressorTypes
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

compressorTypes

A pointer to handle where the list of video compression formats should be returned.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function returns a copy of the list of video compression formats previously passed to `SGSetUserVideoCompressorList` (page 2215). If no video compression formats have been set, it sets the `compressorTypes` handle to `NIL`. The caller of this routine is responsible for disposing of the returned handle.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SGGetUseScreenBuffer

Determines whether a video channel is allowed to use an offscreen buffer.

```
ComponentResult SGGetUseScreenBuffer (
   SGChannel c,
   Boolean *useScreenBuffer
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

useScreenBuffer

A pointer to a Boolean value. The channel component sets this field to TRUE if it draws directly to the screen, or FALSE if it can draw to an offscreen buffer.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function can be called by a sequence grabber client to determine whether or not a video channel can use an offscreen buffer. Some video capture hardware can only capture to the screen.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGGetVideoBottlenecks

Determines the callback functions that have been assigned to a video channel.

```
ComponentResult SGGetVideoBottlenecks (
   SGChannel c,
   VideoBottles *vb
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

vb

A pointer to a `VideoBottles` structure. This function sets the fields of that structure to indicate the callback functions that have been assigned to this video channel. You must set the `procCount` field in the `VideoBottles` structure to 9.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
BrideOfMungGrab
DigitizerShell
Sequence Grabbing

Declared In
`QuickTimeComponents.h`

SGGetVideoCompressor

Determines a channel's current image compression parameters.

```
ComponentResult SGGetVideoCompressor (
   SGChannel c,
   short *depth,
   CompressorComponent *compressor,
   CodecQ *spatialQuality,
   CodecQ *temporalQuality,
   long *keyFrameRate
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

depth

A pointer to a field that is to receive the depth at which the image is likely to be viewed. Image compressor components may use the depth as an indication of the color or grayscale resolution of the compressed images. Return the depth value currently in use by your channel component. If this parameter is set to `NIL`, the sequence grabber component is not interested in this information.

compressor

A pointer to a field that is to receive an image compressor identifier. Return the identifier that corresponds to the image compressor your channel is using. If this parameter is set to `NIL`, the sequence grabber component is not interested in this information.

spatialQuality

A pointer to a field that is to receive the desired compressed image quality. Return the current quality value. If this parameter is set to `NIL`, the sequence grabber component is not interested in this information. See these constants:

`codecMinQuality`

`codecLowQuality`

`codecNormalQuality`

`codecHighQuality`

`codecMaxQuality`

`codecLosslessQuality`

temporalQuality

A pointer to a field that is to receive the desired temporal quality of the sequence. This parameter governs the level of compression you desire with respect to information between successive frames in the sequence. Return the current temporal quality value. If this parameter is set to `NIL`, the sequence grabber component is not interested in this information.

keyFrameRate

A pointer to a field that is to receive the maximum number of frames allowed between key frames. Key frames provide points from which a temporally compressed sequence may be decompressed. This value controls the frequency at which the image compressor places key frames into the compressed sequence. Return the current key frame rate. If this parameter is set to `NIL`, the sequence grabber component is not interested in this information.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGGetVideoCompressorType

Determines the type of image compression that is being applied to a channel's video data.

```
ComponentResult SGGetVideoCompressorType (
   SGChannel c,
   OSType *compressorType
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

compressorType

A pointer to a field that is to receive information about the type of image compression to use. Return a value (see below) that corresponds to one of the image-compression types supported by the Image Compression Manager. You should use `GetCodecNameList` (page 652) to retrieve these names, so that your application can take advantage of new compressor types that may be added in the future. See these constants:

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGGetVideoDigitizerComponent

Determines the video digitizer component that is providing source video to a video channel component.

```
ComponentInstance SGGetVideoDigitizerComponent (
   SGChannel c
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

Return Value
A component instance that identifies the connection between your video channel component and its video digitizer component. If your video channel component does not use a video digitizer component, set this returned value to `NIL`.

Discussion
This function allows the sequence grabber component to determine the video digitizer component that is providing source video to your video channel component. For example, the sequence grabber component can use this function to obtain access to the video digitizer component so that the grabber component can set the digitizer's parameters.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
BrideOfMungGrab

DigitizerShell

Declared In
`QuickTimeComponents.h`

SGGetVideoRect

Determines the portion of the source video image that is to be captured.

```
ComponentResult SGGetVideoRect (
   SGChannel c,
   Rect *r
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

r

A pointer to a `Rect` structure that is to receive the dimensions of the rectangle that defines the portion of the source video image your component is going to capture.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
hacktv

hacktv.win

Declared In
`QuickTimeComponents.h`

SGGrabCompressComplete

Provides the default behavior for your grab-compress-complete function.

```
ComponentResult SGGrabCompressComplete (
   SGChannel c,
   UInt8 *queuedFrameCount,
   SGCompressInfo *ci,
   TimeRecord *tr
);
```

Parameters

c
: The connection identifier for the channel for this operation. The sequence grabber provides this value to your grab-compress-complete function.

queuedFrameCount
: A pointer to the number of queued frames yet to be done. 0 means no frames. Some VDIGs may return 2 even if more than 2 frames are available, and some will return 1 if any number more than 0 are available.

ci
: A pointer to an `SGCompressInfo` structure. When the operation is complete, the function fills in this structure with information about the compression operation.

tr
: A pointer to a `TimeRecord` structure. When the operation is complete, the function uses this structure to indicate when the frame was grabbed.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
By setting the `SGGrabCompressCompleteBottleProc` callback and calling this function, your application can determine how many frames are currently queued in the VDIG, which can be useful for real-time processing.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
BrideOfMungGrab

Declared In
`QuickTimeComponents.h`

SGGrabFrame

Provides the default behavior for your grab function.

```
ComponentResult SGGrabFrame (
   SGChannel c,
   short bufferNum
);
```

Parameters

c

The reference that identifies the channel for this operation. The sequence grabber component provides this value to your grab function.

bufferNum

Identifies the buffer. The sequence grabber component provides this value to your grab function.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SGGrabFrameComplete

Provides the default behavior for your grab-complete function.

```
ComponentResult SGGrabFrameComplete (
   SGChannel c,
   short bufferNum,
   Boolean *done
);
```

Parameters

c

The reference that identifies the channel for this operation. The sequence grabber provides this value to your grab-complete function.

bufferNum

Identifies the buffer. The sequence grabber provides this value to your grab-complete function.

done

A pointer to a Boolean value. The function sets this value to TRUE if the capture is complete, and sets it to FALSE if the capture is incomplete. The sequence grabber provides this pointer to your grab-complete function.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code
DigitizerShell

Sequence Grabbing

Declared In
`QuickTimeComponents.h`

SGGrabPict

Lets your application obtain a Picture structure from a sequence grabber component.

```
ComponentResult SGGrabPict (
   SeqGrabComponent s,
   PicHandle *p,
   const Rect *bounds,
   short offscreenDepth,
   long grabPictFlags
);
```

Parameters

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` or `OpenComponent`.

p

A pointer to a field that is to receive a handle to the `Picture` structure. If the function cannot create the structure, it sets this handle to `NIL`.

bounds

A pointer to the boundary region for the `Picture` structure. By default, this rectangle lies in the current graphics port. If you set the `grabPictOffScreen` flag in the `grabPictFlags` parameter to 1, the sequence grabber places the structure in an offscreen graphics world. In this case, the rectangle is interpreted in that offscreen world.

offscreenDepth

The pixel depth for the offscreen graphics world. This parameter is typically set to 0, which chooses the best available depth. If you set the `grabPictOffScreen` flag in the `grabPictFlags` parameter to 1, the sequence grabber places the `Picture` structure in an offscreen graphics world. You specify the pixel depth of this offscreen graphics world with this parameter. If you are displaying the `picture`, this parameter is ignored.

grabPictFlags

Contains flags (see below) that control the operation. See these constants:

`grabPictOffScreen`

`grabPictIgnoreClip`

`grabPictCurrentImage`

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
hacktv

hacktv.win
qtcapture
qtcapture.win

Declared In
`QuickTimeComponents.h`

SGHandleUpdateEvent

Requests that a sequence grabber handle an update event.

```
ComponentResult SGHandleUpdateEvent (
   SeqGrabComponent s,
   const EventRecord *event,
   Boolean *handled
);
```

Parameters

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` or `OpenComponent`.

event

A pointer to an `EventRecord` structure.

handled

A pointer to a Boolean that returns TRUE if the event was handled, FALSE otherwise.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGIdle

Provides processing time for sequence grabber components.

```
ComponentResult SGIdle (
   SeqGrabComponent s
);
```

Parameters

s

An instance of the sequence grabber component connected to your channel component. The sequence grabber component provides this value through `SGInitChannel` (page 2167).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion

After starting a preview or record operation, the application calls this function as often as possible. The sequence grabber component then calls your `SGIdle` function. This continues until the calling application stops the operation by calling `SGStop` (page 2223).Your `SGIdle` function reports several status and error conditions by means of its result code. If your component returns a nonzero result code during a record operation, the application should call `SGStop` so that the sequence grabber component can store the data it has collected.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qtcapture

qtcapture.win

Sequence Grabbing

SGDataProcSample

VideoProcessing

Declared In

`QuickTimeComponents.h`

SGInitChannel

Initializes a channel component.

```
ComponentResult SGInitChannel (
   SGChannel c,
   SeqGrabComponent owner
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

owner

Identifies the sequence grabber component that has been connected to your channel component. You should save this value so that your channel component can call the utility functions that are provided by the sequence grabber component.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SGInitialize

Initializes the sequence grabber component.

```
ComponentResult SGInitialize (
   SeqGrabComponent s
);
```

Parameters

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` or `OpenComponent`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Before you can call this function you must establish a connection to the sequence grabber component. Use `OpenDefaultComponent` or `OpenComponent` to establish a component connection, as shown below:

```
// SGInitialize coding example
// See "Discovering QuickTime," page 262
SeqGrabComponent MakeMySequenceGrabber (WindowRef pMacWnd)
{
    SeqGrabComponent    seqGrab =NIL;
    OSErr               nErr =noErr;
    // open the default sequence grabber
    seqGrab =OpenDefaultComponent(SeqGrabComponentType, 0);
    if (seqGrab !=NIL) {
        // initialize the default sequence grabber component
        nErr =SGInitialize(seqGrab);
        if (nErr ==noErr) {
        // set its graphics world to the specified window
            nErr =SGSetGWorld(seqGrab, (CGrafPtr)pMacWnd, NIL);
        }
    }
    if (nErr && (seqGrab !=NIL)) {    // clean up on failure
        CloseComponent(seqGrab);
        seqGrab =NIL;
    }
    return seqGrab;
}
```

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Cocoa - SGDataProc

MovieGWorlds

qtcapture

SGDataProcSample

WhackedTV

Declared In

`QuickTimeComponents.h`

SGNewChannel

Creates a sequence grabber channel and assigns a channel component to the channel.

```
ComponentResult SGNewChannel (
   SeqGrabComponent s,
   OSType channelType,
   SGChannel *ref
);
```

Parameters

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` or `OpenComponent`.

channelType

The type of channel to open (see below). This value corresponds to the `component` subtype value of the channel component. See these constants:

`VideoMediaType`

`SoundMediaType`

ref

A pointer to the `frameChannel` field in the `SeqGrabFrameInfo` structure that is to receive a reference to the channel that is added to the sequence grabber component. If the sequence grabber component successfully locates and connects to an appropriate channel component, the sequence grabber component returns a reference to the channel component into this field.

Return Value

See `Error Codes`. If the sequence grabber component cannot open a connection, it sets the result code to a nonzero value. It returns `noErr` if there is no error.

Discussion

The channel component is responsible for providing digitized data to the sequence grabber component. You specify the type of channel component to be added to the sequence grabber component, as shown in the following sample code:

```
// SGNewChannel coding example
// See "Discovering QuickTime," page 263
void MakeMyGrabChannels (SeqGrabComponent    seqGrab,
                         SGChannel           *sgchanVideo,
                         SGChannel           *sgchanSound,
                         const Rect          *rect,
                         Boolean             bWillRecord)
{
    OSErr           nErr;
    long            lUsage;
    // figure out the usage
    lUsage =seqGrabPreview;                 // always previewing
    if (bWillRecord)
        lUsage |=seqGrabRecord;             // sometimes recording
    // create a video channel
    nErr =SGNewChannel(seqGrab, VideoMediaType, sgchanVideo);
    if (nErr ==noErr) {
        // set boundaries for new video channel
        nErr =SGSetChannelBounds(*sgchanVideo, rect);
        // set usage for new video channel
        if (nErr ==noErr)
            nErr =SGSetChannelUsage(*sgchanVideo, lUsage |
                                            seqGrabPlayDuringRecord);
        if (nErr !=noErr) {
            // clean up on failure
            SGDisposeChannel(seqGrab, *sgchanVideo);
            *sgchanVideo =NIL;
        }
    }
    // create a sound channel
    nErr =SGNewChannel(seqGrab, SoundMediaType, sgchanSound);
    if (nErr ==noErr) {
        // set usage of new sound channel
        nErr =SGSetChannelUsage(*sgchanSound, lUsage);
        if (nErr !=noErr) {
            // clean up on failure
            SGDisposeChannel(seqGrab, *sgchanSound);
            *sgchanSound =NIL;
        }
    }
}
```

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
MovieGWorlds

MungSaver

Sequence Grabbing

SGDataProcSample

WhackedTV

Declared In
`QuickTimeComponents.h`

SGNewChannelFromComponent

Creates a sequence grabber channel and assigns a channel component to the channel.

```
ComponentResult SGNewChannelFromComponent (
   SeqGrabComponent s,
   SGChannel *newChannel,
   Component sgChannelComponent
);
```

Parameters

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` or `OpenComponent`.

newChannel
: A pointer to a channel component that is to receive a reference to the channel that is added to the sequence grabber component. If the sequence grabber component successfully locates and connects to the specified channel component, the sequence grabber component returns a reference to the channel component into this field.

sgChannelComponent
: Identifies the channel component to use. You supply a component ID value to the sequence grabber. The sequence grabber then opens a connection to that channel component and returns your connection ID in the field specified by the `newChannel` parameter. You may obtain a component ID value by calling `FindNextComponent`.

Return Value
See `Error Codes`. If the sequence grabber component cannot open a connection, it sets the result code to a nonzero value. It returns `noErr` if there is no error.

Discussion
This function is similar to `SGNewChannel` (page 2169), except that this function allows you to specify a particular component rather than just a component subtype value.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGNewOutput

Creates a new sequence grabber output.

```
ComponentResult SGNewOutput (
   SeqGrabComponent s,
   Handle dataRef,
   OSType dataRefType,
   long whereFlags,
   SGOutput *sgOut
);
```

Parameters

s
: The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` or `OpenComponent`.

dataRef
: A handle to the destination container.

dataRefType
: The type of data reference; see `Data References`. If the data reference is an alias, you must set the parameter to `rAliasType`.

whereFlags

Flags (see below) that control the record operation. You must set either `seqGrabToDisk` or `seqGrabToMemory` to 1. Set unused flags to 0. See these constants:

`seqGrabToDisk`

`seqGrabToMemory`

`seqGrabDontUseTempMemory`

`seqGrabAppendToFile`

`seqGrabDontAddMovieResource`

`seqGrabDontMakeMovie`

sgOut

A pointer to a sequence grabber output. The sequence grabber component returns an output identifier that you can use with other sequence grabber component functions.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Once you have created the sequence grabber output, you can use `SGSetChannelOutput` (page 2191) to assign the output to a sequence grabber channel.

Version Notes

A sequence grabber output ties a sequence grabber channel to a specified data reference for the output of captured data. If you are capturing to a single movie file, you can continue to use `SGSetDataOutput` (page 2197) or `SGSetDataRef` (page 2199) to specify the sequence grabber's destination. However, if you want to capture movie data into several different files or data references, you must use sequence grabber outputs to do so. Even if you are using outputs, you must still use `SGSetDataOutput` or `SGSetDataRef` to identify where the sequence grabber should create the movie resource. You are responsible for creating outputs, assigning them to sequence grabber channels, and disposing of them when you are done.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

hacktv

hacktv.win

qtcapture

qtcapture.win

Declared In

`QuickTimeComponents.h`

SGPanelCanRun

Lets a sequence grabber component determine whether a panel component can work with the current sequence grabber channel component.

```
ComponentResult SGPanelCanRun (
   SeqGrabComponent s,
   SGChannel c
);
```

Parameters

s

Identifies the sequence grabber component's connection to your panel component. See `SGPanelSetGrabber` (page 2181).

c

The connection identifier for the channel for this operation. The sequence grabber component provides you with a connection to the channel component in question. You must determine whether your panel component can operate with this channel component and its associated channel hardware.

Return Value

If your component can work with the specified channel, return a result code of `noErr`. Otherwise, return an appropriate sequence grabber or sequence grabber channel component result code. See `Error Codes`.

Discussion

Set the `channelFlagHasDependency` flag in the `ComponentDescription` structure of your sequence grabber panel component to cause the sequence grabber component to call this function. Your component should query the channel component to determine whether you can operate with it. You may want to use channel component functions to determine the characteristics of the digitization source attached to the channel.

Special Considerations

If your panel component can only support a limited number of connections, you should regulate the number of active connections through this function. Return a nonzero result code to indicate to the sequence grabber that your panel component cannot support the current connection.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SGPanelEvent

Lets a component receive and process dialog events.

```
ComponentResult SGPanelEvent (
   SeqGrabComponent s,
   SGChannel c,
   DialogRef d,
   short itemOffset,
   const EventRecord *theEvent,
   short *itemHit,
   Boolean *handled
);
```

Parameters

s

Identifies the sequence grabber component's connection to your panel component. See `SGPanelSetGrabber` (page 2181).

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

d

A dialog pointer identifying the settings dialog box.

itemOffset

The offset to your panel's first item in the dialog box.

theEvent

A pointer to an `EventRecord` structure. This structure contains information identifying the nature of the event.

itemHit

A pointer to a field that is to receive the item number in cases where your component handles the event. The number returned is an absolute, not a relative number, so it must be offset by the `itemOffset` parameter.

handled

A pointer to a Boolean value. Set this Boolean value to TRUE if you handle the event; set it to FALSE if you do not.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGPanelGetDitl

Lets a sequence grabber component determine the dialog items managed by your panel component.

```
ComponentResult SGPanelGetDitl (
   SeqGrabComponent s,
   Handle *ditl
);
```

Parameters

s

Identifies the sequence grabber component's connection to your panel component. See `SGPanelSetGrabber` (page 2181).

ditl

A pointer to a handle provided by the sequence grabber component. Your component returns the item list in this handle. Your component should resize this handle as appropriate. The sequence grabber component will dispose of this handle after retrieving the item list, so make sure that the item list is not stored in a resource.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The sequence grabber uses the information returned by this function to build a sequence grabber settings dialog box for the user. The sequence grabber component will open your resource file before calling this function, unless you have instructed the sequence grabber component not to open your resource file by setting the `channelFlagDontOpenResFile` flag in your your panel component's `ComponentDescription` structure.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SGPanelGetDITLForSize

Returns user interface elements that fit within a specified size panel.

```
ComponentResult SGPanelGetDITLForSize (
   SeqGrabComponent s,
   Handle *ditl,
   Point *requestedSize
);
```

Parameters

s

An instance of the sequence grabber component connected to your channel component. The sequence grabber component provides this value through `SGInitChannel` (page 2167).

ditl

A pointer to a handle provided by the sequence grabber component. Your panel component returns the dialog item list in this handle. Your component should resize this handle as appropriate. The sequence grabber component will dispose of this handle after retrieving the item list, so make sure that the item list is not stored in a resource.

requestedSize

The size of the panel, or constants (see below). The sequence grabber will interpolate the panel elements between the two sizes if just the constants are returned. See these constants:

`kSGSmallestDITLSize`

`kSGLargestDITLSize`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This routine is used to retrieve user interface elements that fit within a specified size panel. If it is not implemented, the sequenced grabber will assume that your panel does not have resizable user interface elements.

Version Notes

Introduced in QuickTime 6.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`QuickTimeComponents.h`

SGPanelGetSettings

Retrieves a panel's current settings for a sequence grabber component.

```
ComponentResult SGPanelGetSettings (
   SeqGrabComponent s,
   SGChannel c,
   UserData *ud,
   long flags
);
```

Parameters

s

Identifies the sequence grabber component's connection to your panel component. See `SGPanelSetGrabber` (page 2181).

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

ud

A pointer to a `UserDataRecord` structure. Your component is responsible for creating a new structure and returning it by means of this pointer. Your component is not responsible for disposing of the structure. These settings may be stored as part of a larger sequence grabber configuration and may be stored for a long period of time. Therefore, you should not store values that may change without your knowledge (such as component ID or connection values). You are free to format the data in user data items any way you desire.

flags

Reserved for future use.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Make sure your component can retrieve the settings information from the user data item when this function is called. You may choose to format the data in such a way that other components can parse it easily, thus allowing your component to operate with other panel components.

Special Considerations

You create new user data items by calling `NewUserData` (page 1415). You may then use other Movie Toolbox functions to manipulate the user data items.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SGPanelGetTitle

Gets the displayed title of a sequence grabber panel.

```
ComponentResult SGPanelGetTitle (
   SeqGrabComponent s,
   Str255 title
);
```

Parameters

s

Identifies the sequence grabber component's connection to your panel component. See `SGPanelSetGrabber` (page 2181).

title

A string containing the panel's title.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SGPanelInstall

Installs added items in a sequence grabber settings dialog box before the dialog box is displayed to the user.

```
ComponentResult SGPanelInstall (
   SeqGrabComponent s,
   SGChannel c,
   DialogRef d,
   short itemOffset
);
```

Parameters

s

Identifies the sequence grabber component's connection to your panel component. See `SGPanelSetGrabber` (page 2181).

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

d

A dialog pointer identifying the settings dialog box. Your component may use this value to manage its part of the dialog box.

itemOffset

The offset to your panel's first item in the dialog box. Because sequence grabber components build your dialog items into a larger dialog box containing other items, this value may be different each time your panel component is installed; do not rely on it being the same.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

A sequence grabber component calls this function just before displaying the dialog box to the user. The sequence grabber provides you with information identifying the channel that your panel is to configure, the dialog box, and the offset of your panel's items into the dialog box. You may use this opportunity to set default dialog values or to initialize your control values.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SGPanelItem

Receives and processes mouse clicks in the sequence grabber settings dialog box.

```
ComponentResult SGPanelItem (
   SeqGrabComponent s,
   SGChannel c,
   DialogRef d,
   short itemOffset,
   short itemNum
);
```

Parameters

s

Identifies the sequence grabber component's connection to your panel component. See `SGPanelSetGrabber` (page 2181).

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

d

A dialog pointer identifying the settings dialog box.

itemOffset

The offset to your panel's first item in the dialog box.

itemNum

The item number of the dialog item selected by the user. The sequence grabber provides an absolute item number. It is your responsibility to adjust this value to account for the offset to your panel's first item in the dialog box.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

A sequence grabber component calls this function whenever the user clicks an item in the settings dialog box. Your component may then perform whatever processing is appropriate, depending upon the item number.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SGPanelRemove

Removes a panel from the sequence grabber settings dialog box.

```
ComponentResult SGPanelRemove (
   SeqGrabComponent s,
   SGChannel c,
   DialogRef d,
   short itemOffset
);
```

Parameters

s

Identifies the sequence grabber component's connection to your panel component. See `SGPanelSetGrabber` (page 2181).

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

d

A dialog pointer identifying the settings dialog box.

itemOffset

The offset to your panel's first item in the dialog box.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

A sequence grabber component calls this function just before removing your items from the settings dialog box. The sequence grabber provides you with information identifying the channel your panel is to configure, the dialog box, and the offset of your panel's items into the dialog box. You may use this opportunity to save any changes you may have made to the dialog box or to retrieve the contents of text items.

Special Considerations

If the sequence grabber opened your resource file, it will still be open when it calls this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SGPanelSetEventFilter

Sets the event filter callback for a sequence grabber panel component.

```
ComponentResult SGPanelSetEventFilter (
   SeqGrabComponent s,
   SGModalFilterUPP proc,
   long refCon
);
```

Parameters

s
: Identifies the sequence grabber component's connection to your panel component. See `SGPanelSetGrabber` (page 2181).

proc
: An `SGModalFilterProc` callback.

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

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGPanelSetGrabber

Identifies a sequence grabber component to a panel component.

```
ComponentResult SGPanelSetGrabber (
   SeqGrabComponent s,
   SeqGrabComponent sg
);
```

Parameters

s
: Identifies the sequence grabber component's connection to your panel component.

sg
: Identifies a connection to the sequence grabber component that is using your panel component. Your component may use this connection to call sequence grabber component functions.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
A sequence grabber component calls this function in order to identify itself to your panel component. Your component can use the provided connection to call sequence grabber functions, either to determine the characteristics of the current capture operation or to alter those characteristics.

Special Considerations

This is typically the first function a sequence grabber component calls after opening your panel component.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGPanelSetResFile

Lets the sequence grabber pass a resource file's reference number.

```
ComponentResult SGPanelSetResFile (
   SeqGrabComponent s,
   short resRef
);
```

Parameters

s

Identifies the sequence grabber component's connection to your panel component. See `SGPanelSetGrabber` (page 2181).

resRef

A reference number that identifies your component's resource file. After it closes your resource file, the sequence grabber component calls this function and sets this value to 0.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
A sequence grabber component calls this function to pass you your component's resource file reference number. By default, the sequence grabber component opens your component's resource file for you. You can use this reference number to retrieve resources from your resource file. The sequence grabber component also calls this function when it closes your component's resource file. In this case, it sets the `resRef` parameter to 0. The sequence grabber component may close your resource file at any time; you should not count on any particular calling sequence. If you do not want the sequence grabber component to open your resource file, set the `channelFlagDontOpenResFile` flag in your panel component's `ComponentDescription` structure.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGPanelSetSettings

Restores a panel's current settings for a sequence grabber component.

```
ComponentResult SGPanelSetSettings (
   SeqGrabComponent s,
   SGChannel c,
   UserData ud,
   long flags
);
```

Parameters

s

Identifies the sequence grabber component's connection to your panel component. See `SGPanelSetGrabber` (page 2181).

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

ud

Identifies a `UserDataRecord` structure that contains new settings information for your panel. Your component must not dispose of this structure.

flags

Reserved for future use.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Your component originally creates the settings information when the sequence grabber calls `SGPanelGetSettings` (page 2176). The sequence grabber passes this configuration information back to you in the `ud` parameter to this function. Your component should parse the configuration information and use it to establish your panel's current settings. Note that your component may not be able to accommodate the original settings. For example, because the settings may have been stored for some time, the hardware environment may not be able to support the values in the settings. You should try to make your new settings match the original settings as closely as possible. If you cannot get close enough, return an appropriate sequence grabber or sequence grabber channel result code.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGPanelValidateInput

Validates the contents of the user dialog box for a sequence grabber component.

```
ComponentResult SGPanelValidateInput (
   SeqGrabComponent s,
   Boolean *ok
);
```

Parameters

s

Identifies the sequence grabber component's connection to your panel component. See `SGPanelSetGrabber` (page 2181).

ok

A pointer to a Boolean value. Set this value to TRUE if the settings are OK; otherwise, set it to FALSE.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The sequence grabber calls this function when the user clicks the OK button. If the user clicks the Cancel button, the sequence grabber does not call this function. You indicate whether the settings are acceptable by setting the Boolean value referred to by the `ok` parameter. If you set this value to FALSE, the sequence grabber component ignores the OK button in the dialog box.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SGPause

Suspends or restarts a sequence grabber record or preview operation.

```
ComponentResult SGPause (
   SeqGrabComponent s,
   Byte pause
);
```

Parameters

s

An instance of the sequence grabber component connected to your channel component. The sequence grabber component provides this value through `SGInitChannel` (page 2167).

pause

A constant (see below) that instructs your component to suspend or restart the current operation. See these constants:

`seqGrabUnpause`

`seqGrabPause`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Your component should not release any system resources or temporary memory associated with the current operation. You should be ready to restart the operation immediately.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

hacktv

hacktv.win

qtcapture

qtcapture.win

Sequence Grabbing

Declared In

`QuickTimeComponents.h`

SGPrepare

Instructs a sequence grabber to get ready to begin a preview or record operation.

```
ComponentResult SGPrepare (
   SeqGrabComponent s,
   Boolean prepareForPreview,
   Boolean prepareForRecord
);
```

Parameters

s

An instance of the sequence grabber component connected to your channel component. The sequence grabber component provides this value through `SGInitChannel` (page 2167).

prepareForPreview

The sequence grabber component sets this parameter to TRUE to prepare for a preview operation. The sequence grabber component may set both the `prepareForPreview` and `prepareForRecord` parameters to TRUE.

prepareForRecord

The sequence grabber component sets this parameter to TRUE to prepare for a record operation. The sequence grabber component may set both the `prepareForPreview` and `prepareForRecord` parameters to TRUE.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

If you do not call this function before starting a record or preview operation, the sequence grabber component makes these preparations when you start the operation. You cannot call this function after you start a preview or record operation. If you call this function without subsequently starting a record or preview operation, you should call `SGRelease` (page 2186). This allows the sequence grabber component to release any system resources it allocated when you called this function.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
BrideOfMungGrab
Cocoa - SGDataProc
OpenGLCompositorLab
SGDataProcSample
WhackedTV

Declared In
`QuickTimeComponents.h`

SGRelease

Instructs the sequence grabber to release any system resources it allocated when you called SGPrepare.

```
ComponentResult SGRelease (
   SeqGrabComponent s
);
```

Parameters

s

An instance of the sequence grabber component connected to your channel component. The sequence grabber component provides this value through `SGInitChannel` (page 2167).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
You cannot call this function during a record or preview operation.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
AlwaysPreview
BrideOfMungGrab
MungSaver
WhackedTV

Declared In
`QuickTimeComponents.h`

SGSetAdditionalSoundRates

Specifies a list of sound sample rates to be included in the sequence grabber's sound settings dialog box.

```
ComponentResult SGSetAdditionalSoundRates (
   SGChannel c,
   Handle rates
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

rates

A handle containing a list of unsigned 32-bit fixed-point values. The sequence grabber channel determines the number of sample rates contained in the handle, based on the size of the handle. If any of the requested rates are not supported directly by the available sound capture hardware, sound will be captured at one of the available hardware rates and then converted in software to the requested rate.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The sequence grabber channel makes a copy of the additional rates handle, so your application can immediately dispose of it after making this call.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qtcapture

qtcapture.win

Declared In

`QuickTimeComponents.h`

SGSetChannelBounds

Specifies a channel's display boundary rectangle.

```
ComponentResult SGSetChannelBounds (
   SGChannel c,
   const Rect *bounds
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

bounds

A pointer to a `Rect` structure that defines your channel's display boundary rectangle.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
hacktv

hacktv.win

MovieGWorlds

qtcapture

qtcapture.win

Declared In
`QuickTimeComponents.h`

SGSetChannelClip

Sets a channel's clipping region.

```
ComponentResult SGSetChannelClip (
   SGChannel c,
   RgnHandle theClip
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

theClip

A handle to the new clipping region. You should make a copy of this region; the application may dispose of the region immediately.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
AlwaysPreview

Declared In
`QuickTimeComponents.h`

SGSetChannelDevice

Assigns a device to a channel.

```
ComponentResult SGSetChannelDevice (
   SGChannel c,
   StringPtr name
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

name

A pointer to the device's name string. This name is contained in the name field of the appropriate `SGDeviceName` structure in the `SGDeviceListRecord` structure that your channel component returns to the `SGGetChannelDeviceList` (page 2128) function.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SGSetChannelDeviceInput

Undocumented

```
ComponentResult SGSetChannelDeviceInput (
   SGChannel c,
   short inInputNumber
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

inInputNumber

Identifies a device. The value of this field corresponds to the appropriate entry in the device name array defined by the `entry` field of a `SGDeviceListRecord` structure. This value is zero-relative; the first entry has an index number of 0, the second's value is 1, and so on.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 6.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`QuickTimeComponents.h`

SGSetChannelMatrix

Sets a channel's display transformation matrix.

```
ComponentResult SGSetChannelMatrix (
   SGChannel c,
   const MatrixRecord *m
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

m

A pointer to a `MatrixRecord` structure. This parameter is set to `NIL` to select the identity matrix.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

AlwaysPreview

Declared In

`QuickTimeComponents.h`

SGSetChannelMaxFrames

Limits the number of frames that the sequence grabber will capture from a specified channel.

```
ComponentResult SGSetChannelMaxFrames (
   SGChannel c,
   long frameCount
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

frameCount

The maximum number of frames to capture during the preview or record operation. The sequence grabber component sets this parameter to -1 to remove the limit.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

You may use this function only after you have prepared the sequence grabber component for a record operation or during an active record operation.

Special Considerations

Note that sequence grabber components clear this value when you prepare for a record operation.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SGSetChannelOutput

Assigns an output to a channel.

```
ComponentResult SGSetChannelOutput (
   SeqGrabComponent s,
   SGChannel c,
   SGOutput sgOut
);
```

Parameters

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` or `OpenComponent`.

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

sgOut

Identifies the sequence grabber output for this operation. You obtain this identifier by calling `SGNewOutput` (page 2171).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Use this function to assign an output to a channel. One output may be assigned to one or more channels. Note that when you call `SGSetDataRef` (page 2199) or `SGSetDataOutput` (page 2197) the sequence grabber component sets every channel to the specified file or container. If you want to use different outputs, you must use this function to assign the channels appropriately.

Version Notes

A sequence grabber output ties a sequence grabber channel to a specified data reference for the output of captured data. If you are capturing to a single movie file, you can continue to use `SGSetDataOutput` (page 2197) or `SGSetDataRef` (page 2199) to specify the sequence grabber's destination. However, if you want to capture movie data into several different files or data references, you must use sequence grabber outputs to do so. Even if you are using outputs, you must still use `SGSetDataOutput` or `SGSetDataRef` to identify where the sequence grabber should create the movie resource. You are responsible for creating outputs, assigning them to sequence grabber channels, and disposing of them when you are done.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

hacktv

hacktv.win

qtcapture

qtcapture.win

Declared In

`QuickTimeComponents.h`

SGSetChannelPlayFlags

Adjusts the speed and quality with which the sequence grabber displays data from a channel.

```
ComponentResult SGSetChannelPlayFlags (
   SGChannel c,
   long playFlags
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

playFlags

A long integer that contains flags (see below) that influence channel playback. A sequence grabber component must use one of these values. See these constants:

`channelPlayNormal`

`channelPlayFast`

`channelPlayHighQuality`

`channelPlayAllData`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

OpenGLCompositorLab

WhackedTV

Declared In

`QuickTimeComponents.h`

SGSetChannelRefCon

Sets the value of a reference constant that is passed to your callback functions for channel components.

```
ComponentResult SGSetChannelRefCon (
   SGChannel c,
   long refCon
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

refCon

A reference constant value that your component should pass to the callback functions that have been assigned to this channel. Use this parameter to point to a data structure containing any information your callbacks need.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

BrideOfMungGrab

DigitizerShell

Sequence Grabbing

WhackedTV

Declared In

`QuickTimeComponents.h`

SGSetChannelSettings

Configures a sequence grabber channel.

```
ComponentResult SGSetChannelSettings (
   SeqGrabComponent s,
   SGChannel c,
   UserData ud,
   long flags
);
```

Parameters

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` or `OpenComponent`.

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

ud

A `UserDataRecord` structure that contains the configuration information to be used by the channel component.

flags

Reserved for Apple. Set this parameter to 0.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

MungSaver

WhackedTV

Declared In

`QuickTimeComponents.h`

SGSetChannelSettingsStateChanging

Tells a sequence grabber channel of the beginning and end of a group of setting calls.

```
ComponentResult SGSetChannelSettingsStateChanging (
   SGChannel c,
   UInt32 inFlags
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

inFlags

Constants (see below) that determine whether this function is being called at the beginning or end of a series of channel setting calls. See these constants:

`sgSetSettingsBegin`

`sgSetSettingsEnd`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

You can use this call to bracket a group of sequence grabber channel configuration calls, giving downstream components an opportunity to deal with the entire settings change in one operation.

Version Notes

Introduced in QuickTime 6.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`QuickTimeComponents.h`

SGSetChannelUsage

Specifies how a channel is to be used by the sequence grabber component.

```
ComponentResult SGSetChannelUsage (
   SGChannel c,
   long usage
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

usage

Contains flags (see below) specifying how your channel is to be used. The sequence grabber component may set more than one of these flags to 1. It sets unused flags to 0. See these constants:

- `seqGrabRecord`
- `seqGrabPreview`
- `seqGrabPlayDuringRecord`
- `seqGrabLowLatencyCapture`
- `seqGrabAlwaysUseTimeBase`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier. Flags added in QuickTime 6.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

hacktv

hacktv.win

qtcapture

qtcapture.win

SGDataProcSample

Declared In

`QuickTimeComponents.h`

SGSetChannelVolume

Sets a channel's sound volume.

```
ComponentResult SGSetChannelVolume (
   SGChannel c,
   short volume
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

volume

The volume setting of your channel represented as a 16-bit, fixed-point number. The high-order 8 bits contain the integer part of the value; the low-order 8 bits contain the fractional part. Volume values range from -1.0 to 1.0. Negative values play no sound but preserve the absolute value of the volume setting.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

hacktv

hacktv.win

qtcapture

qtcapture.win

Declared In

`QuickTimeComponents.h`

SGSetCompressBuffer

Allows the sequence grabber component to direct your component to create a filter buffer for your video channel.

```
ComponentResult SGSetCompressBuffer (
   SGChannel c,
   short depth,
   const Rect *compressSize
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

depth

The pixel depth of the filter buffer. If the sequence grabber sets this parameter to 0, use the depth of the video buffer, which the sequence grabber sets with `SGSetChannelBounds` (page 2187).

compressSize

A pointer to a `Rect` structure that contains the dimensions of the filter buffer. This buffer should be larger than the destination buffer. To stop filtering the input source video data, the sequence grabber component sets this parameter to `NIL` or it sets the coordinates of this rectangle to 0 (specifying an empty rectangle).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Some video source data may contain unacceptable levels of visual noise or artifacts. One technique for removing this noise is to capture the image and then reduce it in size. During the size reduction process, the noise can be filtered out. Logically, this buffer sits between the source video buffer and the destination rectangle you set with the `SGSetChannelBounds` (page 2187).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SGSetDataOutput

Specifies the movie file and options for a sequence grabber record operation.

```
ComponentResult SGSetDataOutput (
   SeqGrabComponent s,
   const FSSpec *movieFile,
   long whereFlags
);
```

Parameters

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` or `OpenComponent`.

movieFile

A pointer to the `FSSpec` structure that identifies the movie file for this record operation.

whereFlags

Contains flags (see below) that control the record operation. You must set either `seqGrabToDisk` flag or `seqGrabToMemory` to 1. Set unused flags to 0. See these constants:

`seqGrabToDisk`

`seqGrabToMemory`

`seqGrabDontUseTempMemory`

`seqGrabAppendToFile`

`seqGrabDontAddMovieResource`

`seqGrabDontMakeMovie`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
hacktv

hacktv.win

qtcapture

qtcapture.win

Sequence Grabbing

Declared In
`QuickTimeComponents.h`

SGSetDataProc

Specifies a data function for use by the sequence grabber.

```
ComponentResult SGSetDataProc (
   SeqGrabComponent s,
   SGDataUPP proc,
   long refCon
);
```

Parameters

s
: The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` or `OpenComponent`.

proc
: A pointer to your data function. To remove your data function, set this parameter to `NIL`.

refCon
: A reference constant. The sequence grabber provides this value to your data callback. Use this parameter to point to a data structure containing any information your function needs.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Cocoa - SGDataProc

MungSaver

SGDataProcSample

VideoProcessing

WhackedTV

Declared In
`QuickTimeComponents.h`

SGSetDataRef

Specifies the destination data reference for a record operation.

```
ComponentResult SGSetDataRef (
   SeqGrabComponent s,
   Handle dataRef,
   OSType dataRefType,
   long whereFlags
);
```

Parameters

s
: The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` or `OpenComponent`.

dataRef
: A handle to the information that identifies the destination container.

dataRefType
: The type of data reference. If the data reference is an alias, you must set the parameter to `rAliasType`.

whereFlags
: Contains flags (see below) that control the record operation. You must set either `seqGrabToDisk` or `seqGrabToMemory` to 1. Set unused flags to 0. See these constants:

    `seqGrabToDisk`

    `seqGrabToMemory`

    `seqGrabDontUseTempMemory`

    `seqGrabAppendToFile`

    `seqGrabDontAddMovieResource`

    `seqGrabDontMakeMovie`

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function allows you to specify the destination for a record operation using a data reference, and to specify other options that govern the operation. This function is similar to `SGSetDataOutput` (page 2197), and provides you with an alternative way to specify the destination.

Special Considerations

If you are performing a preview operation, you don't need to use this function.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
BrideOfMungGrab

Cocoa - SGDataProc
OpenGLCompositorLab
SGDataProcSample
WhackedTV

Declared In
`QuickTimeComponents.h`

SGSetFlags

Passes control information about the current operation to the sequence grabber component.

```
ComponentResult SGSetFlags (
   SeqGrabComponent s,
   long sgFlags
);
```

Parameters

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` or `OpenComponent`.

sgFlags

Contains a flag (see below) to tell the sequence grabber if you are performing a controlled grab using a frame-addressable source device. See these constants:

`sgFlagControlledGrab`

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Sequence Grabbing

Declared In
`QuickTimeComponents.h`

SGSetFontName

Sets the name of the font to be used to display text for a text channel component.

```
ComponentResult SGSetFontName (
   SGChannel c,
   StringPtr pstr
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

pstr

A pointer to a Pascal string containing the name of the font.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

If the specified font is available on the system, the text channel uses the font to display text. If the specified font is not available, the text channel uses the default system font. For more information about fonts, see *Inside Macintosh: Text*.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SGSetFontSize

Sets the font size to be used to display text for a text channel component.

```
ComponentResult SGSetFontSize (
   SGChannel c,
   short fontSize
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

fontSize

The point size of the font. This value must be a positive integer.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGSetFrameRate

Specifies a video channel's frame rate for recording.

```
ComponentResult SGSetFrameRate (
   SGChannel c,
   Fixed frameRate
);
```

Parameters

c
: The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

frameRate
: The desired frame rate. If this parameter is set to 0, use your channel's default frame rate. Typically, this corresponds to the fastest rate that your channel can support.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
BrideOfMungGrab

Declared In
`QuickTimeComponents.h`

SGSetGWorld

Establishes the graphics port and device for a sequence grabber component.

```
ComponentResult SGSetGWorld (
   SeqGrabComponent s,
   CGrafPtr gp,
   GDHandle gd
);
```

Parameters

s
: An instance of the sequence grabber component connected to your channel component. The sequence grabber component provides this value through `SGInitChannel` (page 2167).

gp
: The destination graphics port, which must be a color graphics port. The sequence grabber component always sets this parameter to a valid value. To use the current graphics port, the parameter is set to `NIL`.

gd

A handle to the destination graphics device. The sequence grabber component always sets this parameter to a valid value.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

You must call this function if you are working with any channels that collect visual data. If you are working only with data that has no visual representation, you do not need to call this function. The sequence grabber component performs this operation implicitly when you call `SGInitialize` (page 2168) and the component uses your application's current graphics port. To set it to a specified window, use code such as the following:

```
// SGSetGWorld coding example
// See "Discovering QuickTime," page 262
SeqGrabComponent MakeMySequenceGrabber (WindowRef pMacWnd)
{
    SeqGrabComponent    seqGrab =NIL;
    OSErr               nErr =noErr;
    // open the default sequence grabber
    seqGrab =OpenDefaultComponent(SeqGrabComponentType, 0);
    if (seqGrab !=NIL) {
        // initialize the default sequence grabber component
        nErr =SGInitialize(seqGrab);
        if (nErr ==noErr) {
        // set its graphics world to the specified window
            nErr =SGSetGWorld(seqGrab, (CGrafPtr)pMacWnd, NIL);
        }
    }
    if (nErr && (seqGrab !=NIL)) {    // clean up on failure
        CloseComponent(seqGrab);
        seqGrab =NIL;
    }
    return seqGrab;
}
```

Special Considerations

You cannot call this function during a record or preview operation, or after you have prepared the sequence grabber component for a record or preview operation by calling `SGPrepare` (page 2185). The window in which the sequence grabber is to draw video frames as defined by this function must be visible before you call `SGPrepare`; otherwise, the sequence grabber does not display the frames properly.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

MovieGWorlds

MungSaver

qtcapture

SGDataProcSample

WhackedTV

Declared In
`QuickTimeComponents.h`

SGSetInstrument

Sets a tone description for a music sequence grabber channel.

```
ComponentResult SGSetInstrument (
   SGChannel c,
   ToneDescription *td
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

td

Pointer to a `ToneDescription` structure.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGSetJustification

Sets the alignment to be used to display text for a text channel component.

```
ComponentResult SGSetJustification (
   SGChannel c,
   short just
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

just

A constant (see below) that represents the text alignment. See these constants:

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
You call this function to specify the alignment to be used for text in a text track. The text channel component justifies text relative to the boundaries of its text box.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGSetMaximumRecordTime

Limits the duration of a record operation

```
ComponentResult SGSetMaximumRecordTime (
   SeqGrabComponent s,
   unsigned long ticks
);
```

Parameters

s
: The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` or `OpenComponent`.

ticks
: The maximum duration for the record operation, in system ticks (sixtieths of a second). Set this parameter to 0 to remove the time limit from the operation.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
By default, there is no time limit on a record operation. If you do not set a limit, a record operation will run until it exhausts the Operating System resources or you call `SGStop` (page 2223). Memory and disk space are the two major limiting factors.

Special Considerations

You must call this function before you start a sequence grabber record operation.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
WhackedTV

Declared In
`QuickTimeComponents.h`

SGSetOutputFlags

Configures an existing sequence grabber output.

```
ComponentResult SGSetOutputFlags (
   SeqGrabComponent s,
   SGOutput sgOut,
   long whereFlags
);
```

Parameters

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` or `OpenComponent`.

sgOut

Identifies the sequence grabber output for this operation. You obtain this identifier by calling `SGNewOutput` (page 2171).

whereFlags

Contains flags (see below) that control the record operation. You must set either `seqGrabToDisk` or `seqGrabToMemory` to 1. Set unused flags to 0. See these constants:

`seqGrabToDisk`

`seqGrabToMemory`

`seqGrabDontUseTempMemory`

`seqGrabAppendToFile`

`seqGrabDontAddMovieResource`

`seqGrabDontMakeMovie`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function lets you configure an existing sequence grabber output.

Version Notes

A sequence grabber output ties a sequence grabber channel to a specified data reference for the output of captured data. If you are capturing to a single movie file, you can continue to use `SGSetDataOutput` (page 2197) or `SGSetDataRef` (page 2199) to specify the sequence grabber's destination. However, if you want to capture movie data into several different files or data references, you must use sequence grabber outputs to do so. Even if you are using outputs, you must still use `SGSetDataOutput` or `SGSetDataRef` to identify where the sequence grabber should create the movie resource. You are responsible for creating outputs, assigning them to sequence grabber channels, and disposing of them when you are done.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SGSetOutputMaximumOffset

Specifies the maximum offset for data written to a specified sequence grabber output.

```
ComponentResult SGSetOutputMaximumOffset (
   SeqGrabComponent s,
   SGOutput sgOut,
   const wide *maxOffset
);
```

Parameters

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` or `OpenComponent`.

sgOut

Identifies the current sequence grabber output. You obtain this identifier by calling `SGNewOutput` (page 2171).

maxOffset

A pointer to the value of the maximum offset for data written to this output.

Return Value

See `Error Codes`. Returns an error if no more outputs are available. Returns `noErr` if there is no error.

Discussion

If an attempt is made to write data beyond the maximum offset, the sequence grabber switches to the next output specified by `SGSetOutputNextOutput` (page 2207). If no more outputs are available, an end-of-file error is returned and recording ends.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SGSetOutputNextOutput

Specifies the order in which sequence grabber outputs should be used.

```
ComponentResult SGSetOutputNextOutput (
   SeqGrabComponent s,
   SGOutput sgOut,
   SGOutput nextOut
);
```

Parameters

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` or `OpenComponent`.

sgOut

The current output to use. When a new output is created, its `nextOut` is set to `NIL`.

nextOut

The next output to be used. To specify that this is the last output, set this value to `NIL`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function should not be called while recording.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGSetPreferredPacketSize

Sets the preferred packet size for the sequence grabber channel component.

```
ComponentResult SGSetPreferredPacketSize (
   SGChannel c,
   long preferredPacketSizeInBytes
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

preferredPacketSizeInBytes

The preferred packet size in bytes.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
This function was added in QuickTime 2.5 to support video conferencing applications.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGSetSettings

Configures a sequence grabber and its channels.

```
ComponentResult SGSetSettings (
   SeqGrabComponent s,
   UserData ud,
   long flags
);
```

Parameters

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` or `OpenComponent`.

ud

A `UserDataRecord` structure that contains the configuration information to be used by the sequence grabber.

flags

Reserved for Apple. Set this parameter to 0.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The sequence grabber disposes of any of its current channels before applying this configuration information. It then opens connections to new channels as appropriate.

Special Considerations

You can restore saved settings by using `NewUserDataFromHandle` (page 1415).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

WhackedTV

Declared In

`QuickTimeComponents.h`

SGSetSettingsSummary

Sets the summary of sequence grabber settings that is displayed in the lower left corner of the sequence grabber dialog.

```
ComponentResult SGSetSettingsSummary (
   SeqGrabComponent s,
   Handle summaryText
);
```

Parameters

s

An instance of the sequence grabber component connected to your channel component. The sequence grabber component provides this value through `SGInitChannel` (page 2167).

summaryText

A handle to the summary text.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This routine supplies a handle (no length byte) that defines a user readable summary of the state of the user's sequence grabber settings.

Version Notes

Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`QuickTimeComponents.h`

SGSetSoundInputDriver

Assigns a sound input device to a sound channel.

```
ComponentResult SGSetSoundInputDriver (
   SGChannel c,
   ConstStr255Param driverName
);
```

Parameters

c
The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

driverName
The name of the sound input device. This is a Pascal string, and it must correspond to a valid sound input device.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGSetSoundInputParameters

Sets various parameters that relate to sound recording.

```
ComponentResult SGSetSoundInputParameters (
   SGChannel c,
   short sampleSize,
   short numChannels,
   OSType compressionType
);
```

Parameters

c
The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

sampleSize
The number of bits in each sound sample. This field is set to 8 for 8-bit sound; it is set to 16 for 16-bit sound.

numChannels
Indicates the number of sound channels to be used by the sound sample. This field is set to 1 for monaural sounds; it is set to 2 for stereo sounds.

compressionType
A constant (see below) that describes the format of the sound data. See these constants:

Return Value
See `Error Codes`. If your sound device cannot support a specified parameter value, return an appropriate Sound Manager result code. Return `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGSetSoundInputRate

Sets the rate at which the sound channel obtains its sound data.

```
ComponentResult SGSetSoundInputRate (
   SGChannel c,
   Fixed rate
);
```

Parameters

c
The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

rate
The rate at which your sound channel is to acquire data. This parameter specifies the number of samples your sound channel is to generate per second. If your sound channel cannot support the specified rate, use the closest available rate that you can support. If this parameter is set to 0, use your default rate.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGSetSoundRecordChunkSize

Controls the amount of sound data in each group of sound samples during a record operation.

```
ComponentResult SGSetSoundRecordChunkSize (
   SGChannel c,
   long seconds
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

seconds

A `Fixed` 16.16 value representing number of seconds of sound data your sound channel component is to work with at a time. Set this parameter to a negative number to specify a fraction of a second. For example, to set the duration to half a second, -0.5 is passed in.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

During record operations, the sequence grabber component and its sound channels work with groups of sound samples, referred to as chunks. By default, each chunk contains two seconds of sound data. Smaller chunks use less memory.

Special Considerations

This function may return a fraction.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SGSetTextBackColor

Sets the background color to be used for the text box.

```
ComponentResult SGSetTextBackColor (
   SGChannel c,
   RGBColor *theColor
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

theColor

A pointer to an `RGBColor` structure that contains the new background color.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion
You call this function to set the background color of a text track. The text channel component uses the specified color as the background of the text box.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGSetTextForeColor

Sets the color to be used to display text.

```
ComponentResult SGSetTextForeColor (
   SGChannel c,
   RGBColor *theColor
);
```

Parameters

`c`
The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

`theColor`
A pointer to an `RGBColor` structure that contains the new text color.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
You call this function to set the text color for a text track.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGSetTextReturnToSpaceValue

Determines whether the text channel component should replace return characters with spaces.

```
ComponentResult SGSetTextReturnToSpaceValue (
   SGChannel c,
   short rettospace
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

rettospace

Set this parameter to TRUE if the text channel should replace return characters with spaces, or FALSE otherwise.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

When you capture text from a closed-caption television source, the text is composed of four lines of text of up to 32 characters each, each line separated by a return character. You can call this function to request that the text channel component replace the return characters with spaces.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SGSettingsDialog

Causes a sequence grabber to display its settings dialog box to the user.

```
ComponentResult SGSettingsDialog (
   SeqGrabComponent s,
   SGChannel c,
   short numPanels,
   ConstComponentListPtr panelList,
   long flags,
   SGModalFilterUPP proc,
   long procRefNum
);
```

Parameters

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` or `OpenComponent`.

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

numPanels

The number of panel components to be listed in the panel component pop-up menu. You specify the panel components with the `panelList` parameter. You may use these parameters to limit the user's choice of panel components. If you set this parameter to 0 and the `panelList` parameter to `NIL`, the sequence grabber lists all available panel components.

panelList

A pointer to an array of component identifiers. The sequence grabber presents only these components in the panel component pop-up menu. You specify the number of identifiers in the array with the `numPanels` parameter. If you set this parameter to `NIL`, the sequence grabber lists all available panel components.

flags

Either set this to 0 or to `seqGrabSettingsPreviewOnly` (see below). See these constants:

`seqGrabSettingsPreviewOnly`

proc

Specifies an `SGModalFilterProc` callback. Because the sequence grabber's settings dialog box is a movable modal dialog box, you must supply an event filter function to process update events in your window.

procRefNum

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

Return Value

See `Error Codes`. If the user clicks OK and the settings are acceptable to the panel and channel components, this function returns a result code of `noErr`.

Discussion

Because the user may change several channel configuration parameters, your application should retrieve new configuration information from the channel so that you can update any values you save, such as the channel's display boundaries or the channel device. In particular, the video rectangle for the channels may have to be adjusted.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

DigitizerShell

hacktv

hacktv.win

qtcapture

WhackedTV

Declared In

`QuickTimeComponents.h`

SGSetUserVideoCompressorList

Specifies the list of video compression formats to be included in the sequence grabber's video settings dialog box.

```
ComponentResult SGSetUserVideoCompressorList (
   SGChannel c,
   Handle compressorTypes
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

compressorTypes

A handle containing a list of `OSType` values indicating which video compression formats should be displayed. See `Codec Identifiers`. The sequence grabber channel determines the number of video compression formats contained in the handle based on the size of the handle.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function lets an application limit the number of video compression formats that will be displayed to the user. For applications using the sequence grabber for a very specific purpose, this function allows inappropriate compression choices to be filtered out.

Special Considerations

The sequence grabber channel makes a copy of the video compression formats handle. Therefore, your application can immediately dispose of the video compression formats handle after making this call.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SGSetUseScreenBuffer

Controls whether a video channel uses an offscreen buffer.

```
ComponentResult SGSetUseScreenBuffer (
   SGChannel c,
   Boolean useScreenBuffer
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

useScreenBuffer

Indicates whether to use an offscreen buffer. If this parameter is set to TRUE, draw directly to the screen. If it is set to FALSE, your channel may use an offscreen buffer. If your channel cannot work with offscreen buffers, ignore this parameter.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Directing a channel to draw offscreen may be useful if you are performing transformations on the data before displaying it (such as blending it with another graphical image).

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGSetVideoBottlenecks

Assigns callback functions to a video channel.

```
ComponentResult SGSetVideoBottlenecks (
   SGChannel c,
   VideoBottles *vb
);
```

Parameters

c

The connection identifier for the video channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

vb

A pointer to a `VideoBottles` structure, which identifies the callback functions to be assigned to the video channel.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
BrideOfMungGrab

DigitizerShell

Sequence Grabbing

Declared In
`QuickTimeComponents.h`

SGSetVideoCompressor

Specifies many of the parameters that control image compression of the video data captured by a video channel.

```
ComponentResult SGSetVideoCompressor (
   SGChannel c,
   short depth,
   CompressorComponent compressor,
   CodecQ spatialQuality,
   CodecQ temporalQuality,
   long keyFrameRate
);
```

Parameters

c

The connection identifier for the video channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

depth

The depth at which the image is likely to be viewed. Image compressors may use this as an indication of the color or grayscale resolution of the compressed images. If the sequence grabber component sets this parameter to 0, let the sequence grabber component determine the appropriate value for the source image. Values of 1, 2, 4, 8, 16, 24, and 32 indicate the number of bits per pixel for color images. Values of 33, 34, 36, and 40 indicate 1-bit, 2-bit, 4-bit, and 8-bit grayscale, respectively, for grayscale images. Your component can determine which depths are supported by a given compressor by examining the `CodecInfo` structure returned by `GetCodecInfo` (page 652).

compressor

The image compressor identifier. The sequence grabber component may specify a particular compressor by setting this parameter to its compressor identifier. You can obtain these identifiers from `GetCodecNameList` (page 652).

spatialQuality

A constant (see below) that defines the desired quality of the compressed image. See these constants:

`codecMinQuality`

`codecLowQuality`

`codecNormalQuality`

`codecHighQuality`

`codecMaxQuality`

`codecLosslessQuality`

temporalQuality

A constant (see below) that defines the desired temporal quality of the sequence. This parameter governs the level of compression the sequence grabber component desires with respect to information in successive frames in the sequence. The sequence grabber component sets this parameter to 0 to prevent the image compressor from applying temporal compression to the sequence.

keyFrameRate

The maximum number of frames allowed between key frames. Key frames provide points from which a temporally compressed sequence may be decompressed. The sequence grabber component uses this parameter to control the frequency with which the image compressor places key frames into the compressed sequence.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGSetVideoCompressorType

Specifies the type of image compression to be applied to captured video images.

```
ComponentResult SGSetVideoCompressorType (
   SGChannel c,
   OSType compressorType
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

compressorType

A constant (see below) that defines the type of image compression to use. You should use `GetCodecNameList` (page 652) to retrieve their names, so that your application can take advantage of new compressor types that may be added in the future. See these constants:

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGSetVideoDigitizerComponent

Assigns a video digitizer component to a video channel.

```
ComponentResult SGSetVideoDigitizerComponent (
   SGChannel c,
   ComponentInstance vdig
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

vdig
A component instance that identifies a connection to a video digitizer component. Your video channel component should use this video digitizer component to obtain its source video data.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGSetVideoRect

Specifies a part of the source video image that is to be captured by a sequence grabber component.

```
ComponentResult SGSetVideoRect (
   SGChannel c,
   const Rect *r
);
```

Parameters

c
The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

r
A pointer to the `Rect` structure that defines the portion of the source video image to be captured. This rectangle must lie within the boundaries of the source video boundary rectangle, which the sequence grabber can obtain by calling `SGGetSrcVideoBounds` (page 2154). If you do not use this function to set a source rectangle, the sequence grabber component captures the entire video image, as defined by the source video boundary rectangle.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
You cannot call this function during a record operation.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
BrideOfMungGrab

Declared In
`QuickTimeComponents.h`

SGSortDeviceList

Sorts a device list alphabetically.

```
ComponentResult SGSortDeviceList (
   SeqGrabComponent s,
   SGDeviceList list
);
```

Parameters

s

The component instance that identifies your connection to the sequence grabber component. You obtain this value from `OpenDefaultComponent` or `OpenComponent`.

list

A pointer to a pointer to an `SGDeviceListRecord` structure.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SGSoundInputDriverChanged

Notifies the sequence grabber component whenever you change the configuration of a sound channel's sound input device.

```
ComponentResult SGSoundInputDriverChanged (
   SGChannel c
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SGStartPreview

Instructs the sequence grabber to begin processing data from its channels.

```
ComponentResult SGStartPreview (
   SeqGrabComponent s
);
```

Parameters

s

An instance of the sequence grabber component connected to your channel component. The sequence grabber component provides this value through `SGInitChannel` (page 2167).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Your channel component should immediately present the data to the user in the appropriate format, according to your channel's configuration. Display video data in the destination display region; play sound data at the specified volume settings.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
hacktv

hacktv.win

qtcapture

qtcapture.win

Sequence Grabbing

Declared In
`QuickTimeComponents.h`

SGStartRecord

Instructs the sequence grabber component to begin collecting data from its channels.

```
ComponentResult SGStartRecord (
   SeqGrabComponent s
);
```

Parameters

s

An instance of the sequence grabber component connected to your channel component. The sequence grabber component provides this value through `SGInitChannel` (page 2167).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
BrideOfMungGrab
CaptureAndCompressIPBMovie
Cocoa - SGDataProc
SGDataProcSample
WhackedTV

Declared In
`QuickTimeComponents.h`

SGStop

Stops a preview or record operation.

```
ComponentResult SGStop (
   SeqGrabComponent s
);
```

Parameters

s

An instance of the sequence grabber component connected to your channel component. The sequence grabber component provides this value through `SGInitChannel` (page 2167).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
It is dangerous to allow an update event to occur during recording. Many digitizers capture directly to the screen, and an update event will result in data loss.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
BrideOfMungGrab
qtcapture
qtcapture.win
Sequence Grabbing
SGDataProcSample

Declared In
`QuickTimeComponents.h`

SGTransferFrameForCompress

Provides the default behavior for your transfer-frame function.

```
ComponentResult SGTransferFrameForCompress (
   SGChannel c,
   short bufferNum,
   const MatrixRecord *mp,
   RgnHandle clipRgn
);
```

Parameters

c

The reference that identifies the channel for this operation. The sequence grabber component provides this value to your transfer-frame function.

bufferNum

Identifies the buffer. The sequence grabber component provides this value to your transfer-frame function.

mp

A pointer to a `MatrixRecord` structure for the transfer operation. If there is no matrix for the operation, set this parameter to `NIL`.

clipRgn

A handle to a `MacRegion` structure that defines the clipping region for the destination image. This region is defined in the destination coordinate system. If there is no clipping region, set this parameter to `NIL`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SGUpdate

Informs your component about update events, to update its display.

```
ComponentResult SGUpdate (
   SeqGrabComponent s,
   RgnHandle updateRgn
);
```

Parameters

s

An instance of the sequence grabber component connected to your channel component. The sequence grabber component provides this value through `SGInitChannel` (page 2167).

updateRgn

Indicates the part of the window that has been changed.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Applications can determine the part of the window that has been changed by examining the appropriate window record. For example, they may call the sequence grabber in this manner:

```
SGUpdate (theSG, ((WindowPeek)updateWindow)->
updateRgn);
```

Special Considerations

Your application should avoid drawing where the sequence grabber is displaying video. Doing so may cause some video digitizer components to stop displaying video.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
hacktv

hacktv.win

qtcapture

Sequence Grabbing

SGDataProcSample

Declared In
`QuickTimeComponents.h`

SGVideoDigitizerChanged

Notifies the sequence grabber component whenever you change the configuration of a video channel's video digitizer.

```
ComponentResult SGVideoDigitizerChanged (
   SGChannel c
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
It is very important to notify the sequence grabber of any configuration changes you make.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGWriteExtendedMovieData

Allows your channel component to add data to a movie.

```
ComponentResult SGWriteExtendedMovieData (
   SeqGrabComponent s,
   SGChannel c,
   Ptr p,
   long len,
   wide *offset,
   SGOutput *sgOut
);
```

Parameters

`s`
: An instance of the sequence grabber component connected to your channel component. The sequence grabber component provides this value through `SGInitChannel` (page 2167).

`c`
: The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

`p`
: The location of the data to be added to the movie.

`len`
: The number of bytes of data to be added to the movie.

`offset`
: A pointer to a wide integer that receives the offset to the new data in the movie. If the movie is in memory, the returned offset reflects the location the data will have in the movie on a permanent storage device.

`sgOut`
: A pointer to the sequence grabber output to which the data was written.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function differs from `SGWriteMovieData` (page 2227), in two respects: the `offset` parameter has a 64-bit value, and the `sgOut` parameter does not exist in `SGWriteMovieData`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGWriteMovieData

Lets a channel component add data to a movie.

```
ComponentResult SGWriteMovieData (
   SeqGrabComponent s,
   SGChannel c,
   Ptr p,
   long len,
   long *offset
);
```

Parameters

s

An instance of the sequence grabber component connected to your channel component. The sequence grabber component provides this value through `SGInitChannel` (page 2167).

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

p

The location of the data to be added to the movie.

len

The number of bytes of data to be added to the movie.

offset

A pointer to a long integer that is to receive the offset to the new data in the movie. The sequence grabber component returns an offset that is correct in the context of a movie resource, even if the movie data is currently stored in memory. That is, if the movie is in memory, the returned offset reflects the location that the data will have in a movie on a permanent storage device, such as a disk.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SGWriteSamples

Called by a sequence grabber component when it is ready to add recorded data to a movie.

```
ComponentResult SGWriteSamples (
   SGChannel c,
   Movie m,
   AliasHandle theFile
);
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

m

Identifies the movie to which your component should add the captured data. Your component should not make any other changes to the movie identified by this reference. Use `SGWriteMovieData` (page 2227) instead.

theFile

Identifies the movie file. The sequence grabber component provides this alias so that you can supply it to the Movie Toolbox. You should not open this file or write to it directly. Use `SGWriteMovieData` (page 2227) instead.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

Callbacks

SGAddFrameBottleProc

Undocumented

```
typedef ComponentResult (*SGAddFrameBottleProcPtr) (SGChannel c, short bufferNum,
 TimeValue atTime, TimeScale scale, const SGCompressInfo *ci,
long refCon);
```

If you name your function `MySGAddFrameBottleProc`, you would declare it this way:

```
ComponentResult MySGAddFrameBottleProc (
    SGChannel               c,
    short                   bufferNum,
    TimeValue               atTime,
    TimeScale               scale,
    const SGCompressInfo    *ci,
    long                    refCon );
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

bufferNum

A buffer identifier provided by the sequence grabber component.

atTime

Undocumented

scale

The current time scale.

ci

A pointer to a `SGCompressInfo` structure.

refCon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

Return Value

See `Error Codes`. Your callback should return `noErr` if there is no error.

Declared In

`QuickTimeComponents.h`

SGCompressBottleProc

Undocumented

```
typedef ComponentResult (*SGCompressBottleProcPtr) (SGChannel c, short bufferNum,
 long refCon);
```

If you name your function `MySGCompressBottleProc`, you would declare it this way:

```
ComponentResult MySGCompressBottleProc (
    SGChannel     c,
    short         bufferNum,
    long          refCon );
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

bufferNum

A buffer identifier provided by the sequence grabber component.

refCon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

Return Value

See `Error Codes`. Your callback should return `noErr` if there is no error.

Declared In

`QuickTimeComponents.h`

SGCompressCompleteBottleProc

Undocumented

```
typedef ComponentResult (*SGCompressCompleteBottleProcPtr) (SGChannel c, short
bufferNum, Boolean *done, SGCompressInfo *ci, long refCon);
```

If you name your function `MySGCompressCompleteBottleProc`, you would declare it this way:

```
ComponentResult MySGCompressCompleteBottleProc (
    SGChannel         c,
    short             bufferNum,
    Boolean           *done,
    SGCompressInfo    *ci,
    long              refCon );
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

bufferNum

A buffer identifier provided by the sequence grabber component.

done

A pointer to a Boolean; return TRUE if the task was completed, FALSE otherwise.

ci

A pointer to a `SGCompressInfo` structure.

refCon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

Return Value

See `Error Codes`. Your callback should return `noErr` if there is no error.

Declared In

`QuickTimeComponents.h`

SGDataProc

Undocumented

```
typedef OSErr (*SGDataProcPtr) (SGChannel c, Ptr p, long len, long *offset, long
chRefCon, TimeValue time, short writeType, long refCon);
```

If you name your function `MySGDataProc`, you would declare it this way:

```
OSErr MySGDataProc (
    SGChannel    c,
    Ptr          p,
    long         len,
    long         *offset,
    long         chRefCon,
    TimeValue    time,
    short        writeType,
    long          refCon );
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

p

Undocumented

len

Undocumented

offset

Undocumented

chRefCon

Undocumented

time

Undocumented

writeType

Undocumented

refCon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

Return Value

See `Error Codes`. Your callback should return `noErr` if there is no error.

Declared In

`QuickTimeComponents.h`

SGDisplayBottleProc

Undocumented

```
typedef ComponentResult (*SGDisplayBottleProcPtr) (SGChannel c, short bufferNum,
MatrixRecord *mp, RgnHandle clipRgn, long refCon);
```

If you name your function `MySGDisplayBottleProc`, you would declare it this way:

```
ComponentResult MySGDisplayBottleProc (
    SGChannel       c,
    short           bufferNum,
    MatrixRecord    *mp,
    RgnHandle       clipRgn,
    long            refCon );
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

bufferNum

A buffer identifier provided by the sequence grabber component.

mp

Undocumented

clipRgn

Undocumented

refCon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

Return Value

See `Error Codes`. Your callback should return `noErr` if there is no error.

Declared In

`QuickTimeComponents.h`

SGDisplayCompressBottleProc

Undocumented

```
typedef ComponentResult (*SGDisplayCompressBottleProcPtr) (SGChannel c, Ptr dataPtr,
 ImageDescriptionHandle desc, MatrixRecord *mp, RgnHandle clipRgn,
long refCon);
```

If you name your function `MySGDisplayCompressBottleProc`, you would declare it this way:

```
ComponentResult MySGDisplayCompressBottleProc (
    SGChannel                 c,
    Ptr                       dataPtr,
    ImageDescriptionHandle    desc,
    MatrixRecord              *mp,
    RgnHandle                 clipRgn,
    long                      refCon );
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

dataPtr

Undocumented

desc

Undocumented

mp

Undocumented

clipRgn

Undocumented

refCon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

Return Value

See `Error Codes`. Your callback should return `noErr` if there is no error.

Declared In

`QuickTimeComponents.h`

SGGrabBottleProc

Undocumented

```
typedef ComponentResult (*SGGrabBottleProcPtr) (SGChannel c, short bufferNum, long
 refCon);
```

If you name your function `MySGGrabBottleProc`, you would declare it this way:

```
ComponentResult MySGGrabBottleProc (
    SGChannel    c,
    short        bufferNum,
    long         refCon );
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

bufferNum

A buffer identifier provided by the sequence grabber component.

refCon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

Return Value

See `Error Codes`. Your callback should return `noErr` if there is no error.

Declared In

`QuickTimeComponents.h`

SGGrabCompleteBottleProc

Undocumented

```
typedef ComponentResult (*SGGrabCompleteBottleProcPtr) (SGChannel c, short bufferNum,
 Boolean *done, long refCon);
```

If you name your function `MySGGrabCompleteBottleProc`, you would declare it this way:

```
ComponentResult MySGGrabCompleteBottleProc (
    SGChannel    c,
    short        bufferNum,
    Boolean      *done,
    long         refCon );
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

bufferNum

A buffer identifier provided by the sequence grabber component.

done

A pointer to a Boolean; return TRUE if the task was completed, FALSE otherwise.

refCon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

Return Value

See `Error Codes`. Your callback should return `noErr` if there is no error.

Declared In

`QuickTimeComponents.h`

SGGrabCompressCompleteBottleProc

Undocumented

```
typedef ComponentResult (*SGGrabCompressCompleteBottleProcPtr) (SGChannel c, Boolean
 *done, SGCompressInfo *ci, TimeRecord *t, long refCon);
```

If you name your function `MySGGrabCompressCompleteBottleProc`, you would declare it this way:

```
ComponentResult MySGGrabCompressCompleteBottleProc (
    SGChannel          c,
    UInt8              *queuedFrameCount,
    SGCompressInfo     *ci,
    TimeRecord         *t,
    long               refCon );
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from `SGNewChannel` (page 2169) or `SGNewChannelFromComponent` (page 2170).

queuedFrameCount

A pointer to the number of queued frames yet to be done. 0 means no frames. Some VDIGs may return 2 even if more than 2 frames are available, and some will return 1 if any number more than 0 are available.

ci

A pointer to a `SGCompressInfo` structure. When the compression operation is complete, this structure is filled with information about it.

t

A pointer to a `TimeRecord` structure. When the compression operation is complete, this structure is used to indicate what time the frame was grabbed.

refCon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

Return Value

See `Error Codes`. Your callback should return `noErr` if there is no error.

Declared In

`QuickTimeComponents.h`

SGModalFilterProc

Undocumented

```
typedef Boolean (*SGModalFilterProcPtr) (DialogPtr theDialog, const EventRecord
*theEvent, short *itemHit, long refCon);
```

If you name your function `MySGModalFilterProc`, you would declare it this way:

```
Boolean MySGModalFilterProc (
    DialogPtr           theDialog,
    const EventRecord   *theEvent,
    short               *itemHit,
    long                refCon );
```

Parameters

theDialog

A pointer to a dialog box.

theEvent

Undocumented

itemHit

Undocumented

refCon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

Return Value

Return TRUE if the event was handled, FALSE otherwise.

Declared In

`QuickTimeComponents.h`

SGTransferFrameBottleProc

Undocumented

```
typedef ComponentResult (*SGTransferFrameBottleProcPtr) (SGChannel c, short
bufferNum, MatrixRecord *mp, RgnHandle clipRgn, long refCon);
```

If you name your function `MySGTransferFrameBottleProc`, you would declare it this way:

```
ComponentResult MySGTransferFrameBottleProc (
    SGChannel       c,
    short           bufferNum,
    MatrixRecord    *mp,
    RgnHandle       clipRgn,
    long            refCon );
```

Parameters

c

The connection identifier for the channel for this operation. You get this value from SGNewChannel (page 2169) or SGNewChannelFromComponent (page 2170).

bufferNum

A buffer identifier provided by the sequence grabber component.

mp

Undocumented

clipRgn

Undocumented

refCon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

Return Value

See Error Codes. Your callback should return noErr if there is no error.

Declared In

QuickTimeComponents.h

Data Types

ConstComponentListPtr

Represents a type used by the Sequence Grabber API.

```
typedef const Component * ConstComponentListPtr;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

QuickTimeComponents.h

SeqGrabComponent

Represents a type used by the Sequence Grabber API.

```
typedef ComponentInstance SeqGrabComponent;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SeqGrabExtendedFrameInfo

Defines a frame for a sequence grabber component and its sequence grabber channel components.

```
struct SeqGrabExtendedFrameInfo {
     wide          frameOffset;
     long          frameTime;
     long          frameSize;
     SGChannel     frameChannel;
     long          frameRefCon;
     SGOutput      frameOutput;
 };
```

Fields

`frameOffset`

Discussion
Specifies the offset to the sample. Note that this is a 64-bit value.

`frameTime`

Discussion
Specifies the time at which the frame was captured by a sequence grabber channel component. The time value is relative to the data sequence. The channel component must choose a time scale and use it consistently for all sample references.

`frameSize`

Discussion
Specifies the number of bytes in the current sample.

`frameChannel`

Discussion
Identifies the current connection to the channel component.

`frameRefCon`

Discussion
Contains a reference constant for use by the channel component. The channel component uses this constant in any appropriate way; for example, to store a reference to frame differencing information for a time-compressed sequence.

`frameOutput`

Discussion
Identifies the sequence grabber output used to store captured data referenced by the current record.

Discussion
This structure differs from `SeqGrabFrameInfo` in two respects: the `frameOffset` field takes a 64-bit value, and the `frameOutput` field does not exist in `SeqGrabFrameInfo`.

Version Notes
Introduced in QuickTime 4.

Related Functions
`SGAddExtendedFrameReference` (page 2107)

SGGetNextExtendedFrameReference (page 2146)

Declared In
QuickTimeComponents.h

SeqGrabExtendedFrameInfoPtr

Represents a type used by the Sequence Grabber API.

```
typedef SeqGrabExtendedFrameInfo * SeqGrabExtendedFrameInfoPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
QuickTimeComponents.h

SeqGrabFrameInfo

Provides information about a frame for a sequence grabber component and its sequence grabber channel components.

```
struct SeqGrabFrameInfo {
     long         frameOffset;
     long         frameTime;
     long         frameSize;
     SGChannel    frameChannel;
     long         frameRefCon;
 };
```

Fields
frameOffset

Discussion
Specifies the offset to the sample. Your channel component obtains this value from SGWriteMovieData (page 2227).

frameTime

Discussion
Specifies the time at which your channel component captured the frame. This time value is relative to the data sequence. That is, this time is not represented in the context of any fixed time scale. Rather, your channel component must choose and use a time scale consistently for all sample references.

frameSize

Discussion
Specifies the number of bytes in the sample described by the sample reference.

frameChannel

Discussion
Identifies the current connection to your channel.

`frameRefCon`

Discussion
Contains a reference constant for use by your channel component. You can use this value in any way that is appropriate for your channel component. For example, video channel components may use this value to store a reference to frame differencing information for a temporally compressed image sequence.

Discussion
This structure differs from `SeqGrabExtendedFrameInfo` in two respects: the `frameOffset` field takes a 32-bit value, and `SeqGrabExtendedFrameInfo` has a `frameOutput` field.

Related Functions
`SGAddFrameReference` (page 2110)
`SGGetNextFrameReference` (page 2147)

Declared In
`QuickTimeComponents.h`

SeqGrabFrameInfoPtr

Represents a type used by the Sequence Grabber API.

```
typedef SeqGrabFrameInfo * SeqGrabFrameInfoPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGAddFrameBottleUPP

Represents a type used by the Sequence Grabber API.

```
typedef STACK_UPP_TYPE(SGAddFrameBottleProcPtr) SGAddFrameBottleUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGChannel

Represents a type used by the Sequence Grabber API.

```
typedef ComponentInstance SGChannel;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGCompressBottleUPP

Represents a type used by the Sequence Grabber API.

```
typedef STACK_UPP_TYPE(SGCompressBottleProcPtr) SGCompressBottleUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGCompressCompleteBottleUPP

Represents a type used by the Sequence Grabber API.

```
typedef STACK_UPP_TYPE(SGCompressCompleteBottleProcPtr) SGCompressCompleteBottleUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGCompressInfo

Defines the characteristics of a buffer that contains a captured image that has been compressed.

```
struct SGCompressInfo {
     Ptr              buffer;
     unsigned long    bufferSize;
     UInt8            similarity;
     UInt8            reserved;
 };
```

Fields
`buffer`
Discussion
Points to the buffer that contains the compressed image. This pointer must contain a 32-bit clean address.

`bufferSize`
Discussion
Specifies the number of bytes of image data in the buffer.

`similarity`
Discussion
Indicates the relative similarity of this image to the previous image in a sequence. A value of 0 indicates that the current frame is a key frame in the sequence. A value of 255 indicates that the current frame is identical to the previous frame. Values from 1 through 254 indicate relative similarity, ranging from very different (1) to very similar (254).

`reserved`
Discussion
Reserved; set to 0.

Discussion

Callback functions use this structure to exchange information about compressed images. For example, `SGCompressCompleteBottleProc` must format a compression information record whenever a video frame is compressed.

Related Functions

`SGCompressCompleteBottleProc`
`SGCompressFrameComplete` (page 2119)
`SGGrabCompressComplete` (page 2163)
`SGGrabCompressCompleteBottleProc`

Declared In

`QuickTimeComponents.h`

SGDataUPP

Represents a type used by the Sequence Grabber API.

```
typedef STACK_UPP_TYPE(SGDataProcPtr) SGDataUPP;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SGDeviceList

Represents a type used by the Sequence Grabber API.

```
typedef SGDeviceListPtr * SGDeviceList;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SGDeviceListPtr

Represents a type used by the Sequence Grabber API.

```
typedef SGDeviceListRecord * SGDeviceListPtr;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

SGDisplayBottleUPP

Represents a type used by the Sequence Grabber API.

```
typedef STACK_UPP_TYPE(SGDisplayBottleProcPtr) SGDisplayBottleUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGDisplayCompressBottleUPP

Represents a type used by the Sequence Grabber API.

```
typedef STACK_UPP_TYPE(SGDisplayCompressBottleProcPtr) SGDisplayCompressBottleUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGGrabBottleUPP

Represents a type used by the Sequence Grabber API.

```
typedef STACK_UPP_TYPE(SGGrabBottleProcPtr) SGGrabBottleUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGGrabCompleteBottleUPP

Represents a type used by the Sequence Grabber API.

```
typedef STACK_UPP_TYPE(SGGrabCompleteBottleProcPtr) SGGrabCompleteBottleUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGGrabCompressCompleteBottleUPP

Represents a type used by the Sequence Grabber API.

```
typedef STACK_UPP_TYPE(SGGrabCompressCompleteBottleProcPtr)
SGGrabCompressCompleteBottleUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGModalFilterUPP

Represents a type used by the Sequence Grabber API.

```
typedef STACK_UPP_TYPE(SGModalFilterProcPtr) SGModalFilterUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGOutput

Represents a type used by the Sequence Grabber API.

```
typedef SGOutputRecord * SGOutput;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

SGOutputRecord

Contains a sequence grabber output.

```
struct SGOutputRecord {
     long    data[1];
 };
```

Fields
`data`

Discussion
An array of data.

Declared In
`QuickTimeComponents.h`

SGTransferFrameBottleUPP

Represents a type used by the Sequence Grabber API.

```
typedef STACK_UPP_TYPE(SGTransferFrameBottleProcPtr) SGTransferFrameBottleUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

VideoBottles

Identifies the callback functions to be assigned to a sequence grabber channel.

```
struct VideoBottles {
     short                               procCount;
     SGGrabBottleUPP                     grabProc;
     SGGrabCompleteBottleUPP             grabCompleteProc;
     SGDisplayBottleUPP                  displayProc;
     SGCompressBottleUPP                 compressProc;
     SGCompressCompleteBottleUPP         compressCompleteProc;
     SGAddFrameBottleUPP                 addFrameProc;
     SGTransferFrameBottleUPP            transferFrameProc;
     SGGrabCompressCompleteBottleUPP     grabCompressCompleteProc;
     SGDisplayCompressBottleUPP          displayCompressProc;
 };
```

Fields
`procCount`

Discussion
Specifies the number of callback functions that may be identified in the structure. Set this field to 9.

`grabProc`

Discussion
Identifies the `SGGrabBottleProc` function. If you are setting such a function, set this field so that it points to the function's entry point. If you are not setting such a function, set this field to `NIL`.

`grabCompleteProc`

Discussion
Identifies the `SGGrabCompleteBottleProc` function. If you are setting such a function, set this field so that it points to the function's entry point. If you are not setting such a function, set this field to `NIL`.

`displayProc`

Discussion
Identifies the `SGDisplayBottleProc` function. If you are setting such a function, set this field so that it points to the function's entry point. If you are not setting such a function, set this field to `NIL`.

`compressProc`

Discussion
Identifies the `SGCompressBottleProc` function. If you are setting such a function, set this field so that it points to the function's entry point. If you are not setting such a function, set this field to `NIL`.

`compressCompleteProc`

Discussion
Identifies the `SGCompressCompleteBottleProc` function. If you are setting such a function, set this field so that it points to the function's entry point. If you are not setting such a function, set this field to `NIL`.

`addFrameProc`

Discussion
Identifies the `SGAddFrameBottleProc` function. If you are setting such a function, set this field so that it points to the function's entry point. If you are not setting such a function, set this field to `NIL`.

`transferFrameProc`

Discussion
Identifies the `SGTransferFrameBottleProc` function. If you are setting such a function, set this field so that it points to the function's entry point. If you are not setting such a function, set this field to `NIL`.

`grabCompressCompleteProc`

Discussion
Identifies the `SGGrabCompressCompleteBottleProc` function. If you are setting such a function, set this field so that it points to the function's entry point. If you are not setting such a function, set this field to `NIL`.

`displayCompressProc`

Discussion
Identifies the `SGDisplayCompressBottleProc` function. If you are setting such a function, set this field so that it points to the function's entry point. If you are not setting such a function, set this field to `NIL`.

Related Functions
`SGGetVideoBottlenecks` (page 2159)
`SGSetVideoBottlenecks` (page 2217)

Declared In
`QuickTimeComponents.h`

Constants

channelPlayAllData

Constants grouped with channelPlayAllData.

```
enum {
  /*
   * Play flag specifying that the SGChannel should use its default
   * preview/playthru methodology.  Currently it is only used by the
   * VideoMediaType SGChannel.
   */
  channelPlayNormal             = 0,
  /*
   * Play flag specifying that the SGChannel should sacrifice playback
   * quality to achieve the specified playback rate.  Currently it is
   * only used by the VideoMediaType SGChannel.
   */
  channelPlayFast               = 1L << 0,
  /*
   * Play flag specifying that the SGChannel should play its data at
   * the highest possible quality. This option sacrifices playback rate
   * for the sake of image quality. It may reduce the amount of
   * processor time available to other programs in the computer. This
   * option should not affect the quality of the recorded data,
   * however.  Currently it is only used by the VideoMediaType
   * SGChannel.
   */
  channelPlayHighQuality        = 1L << 1,
  /*
   * Play flag specifying that the SGChannel should try to play all of
   * the data it captures, even the data that is stored in offscreen
   * buffers. This option is useful when you want to be sure that the
   * user sees as much of the captured data as possible. The sequence
   * grabber component sets this flag to 1 to play all the captured
   * data. The sequence grabber component may combine this flag with
   * any of the other values for the playFlags parameter.  Currently it
   * is only used by the VideoMediaType SGChannel.
   */
  channelPlayAllData            = 1L << 2,
  /*
   * Play flag specifying that the SGChannel should preview/play raw
   * audio samples just after they are captured from its recording
   * device.  Currently it is only used by the SGAudioMediaType
   * SGChannel.
   */
  channelPlayPreMix             = 1L << 3,
  /*
   * Play flag specifying that the SGChannel should preview/play audio
   * samples just after they are mixed down to the client-specified
   * movie track channel layout.  Currently it is only used by the
   * SGAudioMediaType SGChannel.
   */
  channelPlayPostMix            = 1L << 4,
  /*
   * Play flag specifying that the SGChannel should preview/play audio
   * samples just before they are interleaved/converted/compressed to
   * the client-specified movie track format.  Currently it is only
   * used by the SGAudioMediaType SGChannel.
   */
  channelPlayPreConversion      = 1L << 5,
  /*
   * Play flag specifying that the SGChannel should preview/play audio
   * samples after they have been interleaved/converted/compressed to
   * the client-specified movie track format.  Currently it is only
   * used by the SGAudioMediaType SGChannel.
   */
  channelPlayPostConversion     = 1L << 6
};
```

Declared In
`QuickTimeComponents.h`

SGGrabPict Values

Constants passed to SGGrabPict.

```
enum {
  grabPictOffScreen             = 1,
  grabPictIgnoreClip            = 2,
  grabPictCurrentImage          = 4
};
```

Declared In
`QuickTimeComponents.h`

seqGrabCanMoveWindowWhileRecording

Constants grouped with seqGrabCanMoveWindowWhileRecording.

```
enum {
  seqGrabHasBounds              = 1,
  seqGrabHasVolume              = 2,
  seqGrabHasDiscreteSamples     = 4,
  seqGrabDoNotBufferizeData     = 8,
  seqGrabCanMoveWindowWhileRecording = 16
};
```

Declared In
`QuickTimeComponents.h`

seqGrabAlwaysUseTimeBase

Constants grouped with seqGrabAlwaysUseTimeBase.

```
enum {
  seqGrabRecord                 = 1,
  seqGrabPreview                = 2,
  seqGrabPlayDuringRecord       = 4,
  seqGrabLowLatencyCapture      = 8,    /* return the freshest frame possible, for
 live work (videoconferencing, live broadcast, live image processing) */
  seqGrabAlwaysUseTimeBase      = 16,   /* Tell VDIGs to use TimebaseTime always,
 rather than creating uniform frame durations, for more accurate live sync with
audio */
  seqGrabRecordPreferQualityOverFrameRate = 32 /* quality is more important than
frame rate: client rather drop frame instead of lower quality to achieve full frame
 rate */
};
```

Declared In
`QuickTimeComponents.h`

SGSettingsDialog Values

Constants passed to SGSettingsDialog.

```
enum {
  seqGrabSettingsPreviewOnly    = 1
};
```

Declared In
`QuickTimeComponents.h`

seqGrabAppendToFile

Constants grouped with seqGrabAppendToFile.

```
enum {
  seqGrabToDisk                 = 1,
  seqGrabToMemory               = 2,
  seqGrabDontUseTempMemory      = 4,
  seqGrabAppendToFile           = 8,
  seqGrabDontAddMovieResource   = 16,
  seqGrabDontMakeMovie          = 32,
  seqGrabPreExtendFile          = 64,
  seqGrabDataProcIsInterruptSafe = 128,
  seqGrabDataProcDoesOverlappingReads = 256,
  seqGrabDontPreAllocateFileSize = 512  /* Don't set the size of the file before
capture unless the file has been pre-extended */
};
```

Declared In
`QuickTimeComponents.h`

SGGetPause Values

Constants passed to SGGetPause.

```
enum {
  seqGrabUnpause                = 0,
  seqGrabPause                  = 1,
  seqGrabPauseForMenu           = 3
};
```

Declared In
`QuickTimeComponents.h`

SGAddMovieData Values

Constants passed to SGAddMovieData.

```
enum {
  seqGrabWriteAppend            = 0,
  seqGrabWriteReserve           = 1,
  seqGrabWriteFill              = 2
};
```

Declared In
`QuickTimeComponents.h`

SGGetChannelDeviceList Values

Constants passed to SGGetChannelDeviceList.

```
enum {
  sgDeviceListWithIcons         = (1 << 0),
  sgDeviceListDontCheckAvailability = (1 << 1),
  sgDeviceListIncludeInputs     = (1 << 2)
};
```

Declared In
`QuickTimeComponents.h`

sgFlagAllowNonRGBPixMaps

Constants grouped with sgFlagAllowNonRGBPixMaps.

```
enum {
  sgFlagControlledGrab          = (1 << 0),
  sgFlagAllowNonRGBPixMaps      = (1 << 1)
};
```

Declared In
`QuickTimeComponents.h`

SGSetChannelSettingsStateChanging Values

Constants passed to SGSetChannelSettingsStateChanging.

```
enum {
  sgSetSettingsBegin            = (1 << 0), /* SGSetSettings related set calls
about to start*/
  sgSetSettingsEnd              = (1 << 1) /* Finished SGSetSettings calls. Get
ready to use the new settings*/
};
```

Declared In

`QuickTimeComponents.h`

Video Components Reference for QuickTime

Framework:	Frameworks/QuickTime.framework
Declared in	QuickTimeComponents.h

Overview

Video digitizer components convert video input into digitized color images that are compatible with the graphics system of a computer.

Functions by Task

Controlling Analog Video

`VDGetBlackLevelValue` (page 2276)
    Returns the current black level value.

`VDGetBrightness` (page 2277)
    Returns the current brightness value.

`VDGetContrast` (page 2281)
    Returns the current contrast value.

`VDGetHue` (page 2287)
    Returns the current hue value.

`VDGetInputGammaValue` (page 2291)
    Returns the current gamma values.

`VDGetSaturation` (page 2300)
    Returns the current saturation value.

`VDGetSharpness` (page 2301)
    Returns the current sharpness value.

`VDGetVideoDefaults` (page 2305)
    Returns the recommended values for many of the analog video parameters that may be set by applications.

`VDGetWhiteLevelValue` (page 2307)
    Returns the current white level value.

`VDSetBlackLevelValue` (page 2318)
    Sets the current black level value.

`VDSetBrightness` (page 2318)
Sets the current brightness value.

`VDSetContrast` (page 2322)
Sets the current contrast value.

`VDSetHue` (page 2326)
Sets the current hue value.

`VDSetInputGammaValue` (page 2329)
Sets the gamma values.

`VDSetSaturation` (page 2336)
Sets the saturation value.

`VDSetSharpness` (page 2337)
Sets the sharpness value.

`VDSetWhiteLevelValue` (page 2338)
Sets the white level value.

Controlling Color

`VDGetCLUTInUse` (page 2278)
Obtains the color lookup table used by a video digitizer component.

`VDGetDMADepths` (page 2285)
Determines which pixel depths a digitizer supports.

`VDGetInputColorSpaceMode` (page 2289)
Determines whether a digitizer is operating in color or grayscale mode.

`VDSetInputColorSpaceMode` (page 2328)
Chooses between color and grayscale digitized video.

`VDUseThisCLUT` (page 2340)
Specifies the lookup table for color digitization.

Controlling Compressed Source Devices

`VDCompressDone` (page 2273)
Determines whether the video digitizer has finished digitizing and compressing a frame of image data.

`VDCompressOneFrameAsync` (page 2274)
Instructs the video digitizer to digitize and compress a single frame of image data.

`VDGetCompressionTime` (page 2279)
Confirms or quantifies a video digitizer's compression settings.

`VDGetCompressionTypes` (page 2280)
Determines the image-compression capabilities of the video digitizer.

`VDGetImageDescription` (page 2288)
Retrieves an ImageDescription structure from a video digitizer.

`VDGetSoundInputSource` (page 2302)
Instructs your video digitizer component to return the sound input source associated with a particular video input.

`VDReleaseCompressBuffer` (page 2316)
: Frees a buffer received from VDCompressDone.

`VDResetCompressSequence` (page 2317)
: Forces the video digitizer to insert a key frame into a temporally compressed image sequence.

`VDSetCompression` (page 2320)
: Specifies certain compression parameters.

`VDSetCompressionOnOff` (page 2321)
: Allows an application to start and stop compression by video digitizers that can deliver either compressed or uncompressed image data.

`VDSetDataRate` (page 2323)
: Instructs your video digitizer component to limit the rate at which it delivers compressed, digitized video data.

`VDSetTimeBase` (page 2337)
: Establishes the video digitizer's time coordinate system.

Controlling Digitization

`VDCaptureStateChanging` (page 2272)
: Provides process information from a sequence grabber component to the VDIG.

`VDDone` (page 2275)
: Determines if VDGrabOneFrameAsync is finished with a specific output buffer.

`VDGetDataRate` (page 2282)
: Retrieves information that describes the performance capabilities of a video digitizer.

`VDGetTimeCode` (page 2303)
: Instructs your video digitizer component to return timecode information for the incoming video signal.

`VDGrabOneFrame` (page 2307)
: Instructs the video digitizer component to digitize a single frame of source video.

`VDGrabOneFrameAsync` (page 2308)
: Instructs the video digitizer component to start to digitize asynchronously a single frame of source video.

`VDReleaseAsyncBuffers` (page 2316)
: Releases the buffers that were allocated with VDSetupBuffers.

`VDSetFrameRate` (page 2326)
: Indicates an application's desired frame rate to the video digitizer.

`VDSetPlayThruOnOff` (page 2334)
: Controls continuous digitization.

`VDSetPreferredPacketSize` (page 2335)
: Sets the preferred packet size for video digitizing.

`VDSetupBuffers` (page 2338)
: Defines output buffers for use with asynchronous grabs.

Controlling the Video Output Display Mode

`QTVideoOutputGetDisplayMode` (page 2264)
Returns the current display mode for a video output component.

`QTVideoOutputGetDisplayModeList` (page 2264)
Returns a list of the display modes supported by a video output component.

`QTVideoOutputSetDisplayMode` (page 2270)
Specifies the display mode to be used by a video output component.

Controlling Video Output

`QTVideoOutputBaseSetEchoPort` (page 2258)
Called on the base video output component to inform it about a change in the echo port.

`QTVideoOutputBegin` (page 2259)
Obtains exclusive access to the video hardware controlled by a video output component.

`QTVideoOutputCustomConfigureDisplay` (page 2260)
Displays a custom video configuration dialog box, which can include settings that are specific to the video device controlled by the video output component.

`QTVideoOutputEnd` (page 2261)
Releases access to the video hardware controlled by a video output component.

`QTVideoOutputGetGWorld` (page 2265)
Returns a pointer to the graphics world used by a video output component.

`QTVideoOutputGetGWorldParameters` (page 2266)
Called by the base video output component as part of its implementation of QTVideoOutputGetGWorld.

`QTVideoOutputSetEchoPort` (page 2271)
Specifies a window on the desktop in which to display video sent to the device.

Finding Components Associated With a Video Output

`QTVideoOutputGetClock` (page 2262)
Returns a pointer to the clock component associated with the video output component.

`QTVideoOutputGetIndSoundOutput` (page 2267)
Determines which sound output components are associated with the video output component.

Getting Information About Video Digitizer Components

`VDGetCurrentFlags` (page 2281)
Returns status information about a specified video digitizer component.

`VDGetDeviceNameAndFlags` (page 2283)
Returns the current name and device visibility of a video digitizer.

`VDGetDigitizerInfo` (page 2284)
Returns capability and status information about a specified video digitizer component.

`VDGetUniqueIDs` (page 2304)
: Returns a unique identifier for a particular video digitizer device.

`VDSelectUniqueIDs` (page 2317)
: Selects a video digitizer device by ID.

Registering the Name of Video Output Software

`QTVideoOutputGetClientName` (page 2262)
: Obtains the name of the application or other software that is registered with an instance of a video output component.

`QTVideoOutputGetCurrentClientName` (page 2263)
: Returns the name of the software, if any, that has exclusive access to the video hardware controlled by a video output component.

`QTVideoOutputSetClientName` (page 2270)
: Registers the name of an application or other software with an instance of a video output component.

Saving and Restoring Component Configurations

`QTVideoOutputRestoreState` (page 2268)
: Restores the previously saved state of a video output component.

`QTVideoOutputSaveState` (page 2269)
: Saves state information for an instance of a video output component.

Selecting an Input Source

`VDGetInput` (page 2288)
: Returns data that identifies the currently active input video source.

`VDGetInputFormat` (page 2289)
: Determines the format of the video signal provided by a specified video input source.

`VDGetNumberOfInputs` (page 2297)
: Returns the number of input video sources that a video digitizer component supports.

`VDSetInput` (page 2327)
: Selects the input video source for a video digitizer component.

`VDSetInputStandard` (page 2329)
: Specifies the input signaling standard to digitize.

Selectively Displaying Video

`VDAddKeyColor` (page 2272)
: Adds a key color to a component's list of active key colors.

`VDGetKeyColor` (page 2292)
: Obtains the index value of the active key color.

`VDGetKeyColorRange` (page 2293)
: Obtains the currently defined key color range.

`VDGetMaskandValue` (page 2294)
: Obtains the appropriate alpha channel or blend mask value for a desired level of video blending.

`VDGetMaskPixMap` (page 2295)
: Retrieves the pixel map data for a component's blend mask.

`VDGetNextKeyColor` (page 2296)
: Obtains the index value of the active key colors in cases where the digitizer component supports multiple key colors.

`VDSetKeyColor` (page 2330)
: Sets the key color for video digitizing.

`VDSetKeyColorRange` (page 2331)
: Defines a key color range for video digitizing.

`VDSetMasterBlendLevel` (page 2331)
: Sets the blend level value for the input video signal.

Setting Source Characteristics

`VDGetActiveSrcRect` (page 2276)
: Obtains size and location information for the active source rectangle used by a video digitizer component.

`VDGetDigitizerRect` (page 2285)
: Returns the current digitizer rectangle.

`VDGetMaxSrcRect` (page 2296)
: Returns the maximum source rectangle.

`VDGetVBlankRect` (page 2304)
: Returns the vertical blanking rectangle.

`VDSetDigitizerRect` (page 2324)
: Sets the current video digitizer rectangle.

Setting Video Destinations

`VDGetMaxAuxBuffer` (page 2295)
: Obtains access to buffers that are located on special hardware.

`VDGetPlayThruDestination` (page 2298)
: Obtains information about the current video destination.

`VDPreflightDestination` (page 2314)
: Verifies that a video digitizer component can support a set of destination settings intended for use with VDSetPlayThruDestination.

`VDPreflightGlobalRect` (page 2315)
: Verifies that a video digitizer component can support a set of destination settings intended for use with VDSetPlayThruGlobalRect.

`VDSetPlayThruDestination` (page 2332)
: Establishes the destination settings for a video digitizer component.

`VDSetPlayThruGlobalRect` (page 2333)
Establishes the destination settings for a video digitizer component that is to digitize into a global rectangle.

Video Clipping

`VDClearClipRgn` (page 2273)
Disables all or part of a clipping region that was previously set with VDSetClipRgn.

`VDGetClipState` (page 2278)
Determines whether clipping is enabled.

`VDSetClipRgn` (page 2319)
Defines a clipping region for a video digitizer.

`VDSetClipState` (page 2320)
Controls whether clipping is enabled.

Video Digitizer Utilities

`VDGetFieldPreference` (page 2286)
Determines which field is being used in cases where the image is vertically scaled to half its original size.

`VDGetPLLFilterType` (page 2299)
Determines which phase locked loop (PLL) mode is currently active for a video digitizer.

`VDGetPreferredTimeScale` (page 2300)
Determines a digitizer's preferred time scale.

`VDGetSoundInputDriver` (page 2302)
Retrieves information about a video digitizer's sound input driver.

`VDSetDigitizerUserInterrupt` (page 2324)
Sets custom interrupt functions.

`VDSetFieldPreference` (page 2325)
Specifies which field to use in cases where the vertical scaling is less than half size.

`VDSetPLLFilterType` (page 2334)
Specifies which phase locked loop (PLL) is to be active.

Supporting Functions

`QTVideoOutputCopyIndAudioOutputDeviceUID` (page 2260)
Identifies the audio device being used by a video output component.

`QTVideoOutputGetIndImageDecompressor` (page 2267)
Undocumented

`VDGetInputGammaRecord` (page 2290)
Retrieves a pointer to the active input VDGammaRecord structure for a video digitizer.

`VDGetInputName` (page 2292)
Gets the name of a video input.

`VDGetPreferredImageDimensions` (page 2299)
: Gets the preferred image dimensions for a video digitizer.

`VDIIDCGetCSRData` (page 2309)
: Reads a camera's CSR registers directly.

`VDIIDCGetDefaultFeatures` (page 2310)
: Places atoms in a QuickTime atom container that specify the default capabilities and default state of a camera's IIDC features.

`VDIIDCGetFeatures` (page 2310)
: Places atoms in a QuickTime atom container that specify the current capabilities of a camera and the state of its IIDC features.

`VDIIDCGetFeaturesForSpecifier` (page 2311)
: Places atoms in a QuickTime atom container that specify the current state of a single camera IIDC feature or group of features.

`VDIIDCSetCSRData` (page 2312)
: Writes to a camera's CSR registers directly.

`VDIIDCSetFeatures` (page 2313)
: Changes the state of a camera's IIDC features.

`VDSetDestinationPort` (page 2323)
: Sets the destination port for a video digitizer.

`VDSetInputGammaRecord` (page 2328)
: Changes the active input gamma data structure.

`VDSetPreferredImageDimensions` (page 2335)
: Sets the preferred image dimensions for a video digitizer.

`VDUseSafeBuffers` (page 2339)
: Instructs a video digitizer to use protected buffers.

Functions

QTVideoOutputBaseSetEchoPort

Called on the base video output component to inform it about a change in the echo port.

```
ComponentResult QTVideoOutputBaseSetEchoPort (
   QTVideoOutputComponent vo,
   CGrafPtr echoPort
);
```

Parameters

vo
: The instance of a video output component for this request. Your software obtains this reference when calling `OpenComponent` or `OpenDefaultComponent`.

echoPort
: The window on the computer's desktop in which to display the video.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
SoftVideoOutputComponent

Declared In
`QuickTimeComponents.h`

QTVideoOutputBegin

Obtains exclusive access to the video hardware controlled by a video output component.

```
ComponentResult QTVideoOutputBegin (
   QTVideoOutputComponent vo
);
```

Parameters

vo

The instance of a video output component. Your software obtains this reference when calling `OpenComponent` or `OpenDefaultComponent`.

Return Value
See `Error Codes`. Returns `noErr` if there is no error. If this function returns the `videoOutputInUseErr` result code that indicates that the video hardware is currently in use, your software can get the name of the application or other software that is using the hardware by calling `QTVideoOutputGetCurrentClientName` (page 2263). You can then display an alert to the user that says that the video hardware is in use and specifies the name of the software using the video hardware.

Discussion
When your software calls this function, the video output component acquires exclusive access to the video hardware controlled by the specified video output component or returns the `videoOutputInUseErr` result code if the video hardware is currently in use. If the video hardware is available, the video output component also enables the display mode last set with `QTVideoOutputSetDisplayMode` (page 2270) and enables the video settings, if any, that were most recently specified by the user in a custom video configuration dialog box. If the video output component supports `QTVideoOutputCustomConfigureDisplay` (page 2260), your software can call the function to display a custom video configuration dialog box. When your software no longer needs the video output component, release it by calling `QTVideoOutputEnd` (page 2261).

Special Considerations

If your software needs to change the display mode, it must change it before calling this function. It cannot change the display mode between calls to this function and to `QTVideoOutputEnd` (page 2261).

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Quartz Composer Live DV

Quartz Composer QCTV
SoftVideoOutputComponent

Declared In
`QuickTimeComponents.h`

QTVideoOutputCopyIndAudioOutputDeviceUID

Identifies the audio device being used by a video output component.

```
ComponentResult QTVideoOutputCopyIndAudioOutputDeviceUID (
   QTVideoOutputComponent vo,
   long index,
   CFStringRef *audioDeviceUID
);
```

Parameters

vo
: Video output component whose audio output is being asked about.

index
: Which of video output component's audio outputs is being asked about.

audioDeviceUID
: Returned unique identifier for the audio device. If the UID is `NIL`, the movie is playing to the default device.

Return Value
See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error. Returns `badComponentInstance` if `vo` is not a valid `ComponentInstance`. Returns `badComponentSelector` if `vo` doesn't support this function. Returns `paramErr` if `audioDeviceUID` is `NIL`, or if there is no device with the passed index.

Discussion
The returned `audioDeviceUID` has already been retained for the caller, using standard Core Foundation copy semantics.

Version Notes
Introduced in QuickTime 6.4.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`QuickTimeComponents.h`

QTVideoOutputCustomConfigureDisplay

Displays a custom video configuration dialog box, which can include settings that are specific to the video device controlled by the video output component.

```
ComponentResult QTVideoOutputCustomConfigureDisplay (
   QTVideoOutputComponent vo,
   ModalFilterUPP filter
);
```

Parameters

vo

The instance of a video output component for this request. Your software obtains this reference when calling `OpenComponent` or `OpenDefaultComponent`.

filter

A `ModalFilterProc` callback for the video output component to use for the dialog box. The filter allows the software to process events while the dialog box is displayed.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Your software can determine if a video output component supports this function by calling `ComponentFunctionImplemented` for the component with the routine selector `kQTVideoOutputCustomConfigureDisplaySelect`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

QTVideoOutputEnd

Releases access to the video hardware controlled by a video output component.

```
ComponentResult QTVideoOutputEnd (
   QTVideoOutputComponent vo
);
```

Parameters

vo

The instance of a video output component. Your software obtains this reference when calling `OpenComponent` or `OpenDefaultComponent`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Your software should release access to a video output component as soon as it is done using the video hardware controlled by the component. If you close the instance of a video output component that currently has exclusive access to video hardware, the video output component automatically calls this function to release the hardware.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Quartz Composer Live DV
Quartz Composer QCTV
SoftVideoOutputComponent

Declared In
`QuickTimeComponents.h`

QTVideoOutputGetClientName

Obtains the name of the application or other software that is registered with an instance of a video output component.

```
ComponentResult QTVideoOutputGetClientName (
   QTVideoOutputComponent vo,
   Str255 str
);
```

Parameters

vo
: The instance of a video output component for the request. Your software obtains this reference when it calls `OpenComponent` or `OpenDefaultComponent`.

str
: The name of the application or other software that is registered with the component instance.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

QTVideoOutputGetClock

Returns a pointer to the clock component associated with the video output component.

```
ComponentResult QTVideoOutputGetClock (
   QTVideoOutputComponent vo,
   ComponentInstance *clock
);
```

Parameters

vo
: The instance of a video output component for this request. Your software obtains this reference when calling `OpenComponent` or `OpenDefaultComponent`.

clock

A pointer to the clock component associated with the video output component.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Your software can use the clock component returned by this function to synchronize video and sound for a movie to the rate of the display. To associate the instance of the clock component with a movie, call `SetMovieMasterClock` (page 290). Because a change to the display mode could affect a clock component, your software should call this function only between calls to `QTVideoOutputBegin` (page 2259) and `QTVideoOutputEnd` (page 2261), when it is not possible to change the display mode.

Special Considerations

When your software calls `QTVideoOutputEnd` (page 2261), the video output component disposes of the instance of the clock component returned by this function. Because of this, software that uses the clock to control a movie must reset the clock for the movie to the default clock, by calling `SetMovieMasterClock` (page 290) with `NIL` as the value of the clock component, before calling `QTVideoOutputEnd`.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

QTVideoOutputGetCurrentClientName

Returns the name of the software, if any, that has exclusive access to the video hardware controlled by a video output component.

```
ComponentResult QTVideoOutputGetCurrentClientName (
   QTVideoOutputComponent vo,
   Str255 str
);
```

Parameters

vo

The instance of a video output component for this request. Your software obtains this reference when calling `OpenComponent` or `OpenDefaultComponent`.

str

The name of the software that has exclusive access to the video hardware controlled by a video output component, or a zero-length string if no software currently has access.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

If video hardware is unavailable because other software is using it, your software can inform users by getting the name of the software with this function and displaying the name in an alert box.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

QTVideoOutputGetDisplayMode

Returns the current display mode for a video output component.

```
ComponentResult QTVideoOutputGetDisplayMode (
   QTVideoOutputComponent vo,
   long *displayModeID
);
```

Parameters

vo

The instance of a video output component. Your software obtains this reference when calling `OpenComponent` or `OpenDefaultComponent`.

displayModeID

A pointer to the ID of the current display mode, or 0 if no display mode has been selected. The ID specifies a QT atom of type `kQTVODisplayModeItem` in the QT atom container returned by `QTVideoOutputGetDisplayModeList` (page 2264).

Return Value
See `Error Codes`. Returns `noErr` if there is no error. If this function returns an atom ID of 0, it indicates that no display mode has been selected.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
SoftVideoOutputComponent

Declared In
`QuickTimeComponents.h`

QTVideoOutputGetDisplayModeList

Returns a list of the display modes supported by a video output component.

```
ComponentResult QTVideoOutputGetDisplayModeList (
   QTVideoOutputComponent vo,
   QTAtomContainer *outputs
);
```

Parameters

vo

The instance of a video output component. Your software obtains this reference when calling `OpenComponent` or `OpenDefaultComponent`.

outputs

A pointer to the QT atom container that lists the video modes supported by this component.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

After your software calls this function, it must dispose of the QT atom container returned by the function by calling `QTDisposeAtomContainer` (page 1427).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Fiendishthngs

Declared In

`QuickTimeComponents.h`

QTVideoOutputGetGWorld

Returns a pointer to the graphics world used by a video output component.

```
ComponentResult QTVideoOutputGetGWorld (
   QTVideoOutputComponent vo,
   GWorldPtr *gw
);
```

Parameters

vo

The instance of a video output component for this request. Your software obtains this reference when calling `OpenComponent` or `OpenDefaultComponent`.

gw

A pointer to the graphics world used by the video output component to display images.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

If the pixel format of the graphics world is 1, 2, 4, 8, 16, or 32, your software can use either QuickDraw or QuickTime to draw graphics to it. If the graphics world has any other pixel format, your software must use QuickTime functions draw to it. Your software can pass the pointer returned by this function to the `SetMovieGWorld` (page 290), `DecompressSequenceBegin` (page 621), `DecompressSequenceBeginS` (page 622), `DecompressImage` (page 619), and `FDecompressImage` (page 643) functions.

Your software can call `QTVideoOutputGetGWorld` only between calls to `QTVideoOutputBegin` (page 2259) and `QTVideoOutputEnd` (page 2261). When your software calls `QTVideoOutputEnd`, the video output component automatically disposes of the graphics world. If your software needs to use the graphics world after calling `QTVideoOutputEnd`, it must call this function again after the next time it calls `QTVideoOutputBegin`.

Special Considerations

Your software must not dispose of the graphics world used by a video output component. The video output component automatically disposes of the graphics world when your software calls `QTVideoOutputEnd` (page 2261).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Quartz Composer Live DV

Quartz Composer QCTV

Declared In

`QuickTimeComponents.h`

QTVideoOutputGetGWorldParameters

Called by the base video output component as part of its implementation of QTVideoOutputGetGWorld.

```
ComponentResult QTVideoOutputGetGWorldParameters (
   QTVideoOutputComponent vo,
   Ptr *baseAddr,
   long *rowBytes,
   CTabHandle *colorTable
);
```

Parameters

vo

An instance of your video output component.

baseAddr

The address at which to display pixels. If your video output component does not display pixels, return 0 for this parameter.

rowBytes

The width of each scan line in bytes. If your video output component does not display pixels, return the width of the current display mode.

colorTable

The `ColorTable` structure to be used. If your video output component does not use a color table, return `NIL`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function is not called by applications or other client software.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

QTVideoOutputGetIndImageDecompressor

Undocumented

```
ComponentResult QTVideoOutputGetIndImageDecompressor (
   QTVideoOutputComponent vo,
   long index,
   Component *codec
);
```

Parameters

vo

Undocumented

index

Undocumented

codec

Undocumented

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 5.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

QTVideoOutputGetIndSoundOutput

Determines which sound output components are associated with the video output component.

```
ComponentResult QTVideoOutputGetIndSoundOutput (
   QTVideoOutputComponent vo,
   long index,
   Component *outputComponent
);
```

Parameters

vo

The instance of a video output component for this request. Your software obtains this reference when calling `OpenComponent` or `OpenDefaultComponent`.

index

Specifies which of the sound output components to return. The index of the first component is 1.

outputComponent

A pointer to a sound output component associated with the video output component that is specified by the `index` parameter.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Your software can display sound output components returned by this function in a dialog box and let the user choose which outputs to use for movie playback.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

QTVideoOutputRestoreState

Restores the previously saved state of a video output component.

```
ComponentResult QTVideoOutputRestoreState (
   QTVideoOutputComponent vo,
   QTAtomContainer state
);
```

Parameters

vo

The instance of a video output component for this request. Your software obtains this reference when calling `OpenComponent` or `OpenDefaultComponent`.

state

A QT atom container, returned earlier by `QTVideoOutputSaveState` (page 2269), that contains state information for the video output component.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

If your software saves state information to disk, it must read the QT atom container structure from disk before calling this function. When your software restores state information for a video output component, the current display mode may change. Because of this, your software must call this function before calling `QTVideoOutputBegin` (page 2259).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

QTVideoOutputSaveState

Saves state information for an instance of a video output component.

```
ComponentResult QTVideoOutputSaveState (
   QTVideoOutputComponent vo,
   QTAtomContainer *state
);
```

Parameters

vo

The instance of a video output component for this request. Your software obtains this reference when calling `OpenComponent` or `OpenDefaultComponent`.

state

A pointer to complete information about the video output component's current configuration.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

When your software saves state information for an instance of a video output component, it can restore this information when reconnecting to the component by calling `QTVideoOutputRestoreState` (page 2268).

Special Considerations

When your software calls this function, it must dispose of the QT atom container returned by the function by calling `QTDisposeAtomContainer` (page 1427).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

QTVideoOutputSetClientName

Registers the name of an application or other software with an instance of a video output component.

```
ComponentResult QTVideoOutputSetClientName (
   QTVideoOutputComponent vo,
   ConstStr255Param str
);
```

Parameters

vo

The instance of a video output component for the request. Your software obtains this reference when it calls `OpenComponent` or `OpenDefaultComponent`.

str

The name of the application or other software to be registered.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The name you specify with this function can later be used by `QTVideoOutputGetCurrentClientName` (page 2263) to specify which software has exclusive access to the video output device controlled by the component.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Quartz Composer Live DV

Quartz Composer QCTV

Declared In

`QuickTimeComponents.h`

QTVideoOutputSetDisplayMode

Specifies the display mode to be used by a video output component.

```
ComponentResult QTVideoOutputSetDisplayMode (
   QTVideoOutputComponent vo,
   long displayModeID
);
```

Parameters

vo

The instance of a video output component for the request. Your software obtains this reference when calling `OpenComponent` or `OpenDefaultComponent`.

displayModeID

The ID of the display mode to use. The ID specifies a QT atom of type `kQTVODisplayModeItem` in the QT atom container returned by `QTVideoOutputGetDisplayModeList` (page 2264).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion
When software changes the display mode with this function, the change does not take effect until the next time the software calls `QTVideoOutputBegin` (page 2259) for the video output component. This lets the software change other output settings before displaying the video.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Quartz Composer Live DV

Quartz Composer QCTV

Declared In
`QuickTimeComponents.h`

QTVideoOutputSetEchoPort

Specifies a window on the desktop in which to display video sent to the device.

```
ComponentResult QTVideoOutputSetEchoPort (
   QTVideoOutputComponent vo,
   CGrafPtr echoPort
);
```

Parameters
vo

The instance of a video output component for this request. Your software obtains this reference when calling `OpenComponent` or `OpenDefaultComponent`.

echoPort

The window on the computer's desktop in which to display the video.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
When your software sends video to the window you specify, the video is both displayed in the window and sent to the normal output of the video output device. When an output device can display video both on an external video display and in a window on a computer's desktop, the video displayed on the desktop is often at a smaller size and/or lower frame rate.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

VDAddKeyColor

Adds a key color to a component's list of active key colors.

```
VideoDigitizerError VDAddKeyColor (
   VideoDigitizerComponent ci,
   long *index
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

index

A pointer to the color to add to the key color list. The value of the `index` field corresponds to a color in the current color lookup table.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

VDCaptureStateChanging

Provides process information from a sequence grabber component to the VDIG.

```
VideoDigitizerError VDCaptureStateChanging (
   VideoDigitizerComponent ci,
   UInt32 inStateFlags
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

inStateFlags

Constants (see below) that tell the VDIG what is about to happen. See these constants:

`vdFlagCaptureStarting`

`vdFlagCaptureStopping`

`vdFlagCaptureIsForPreview`

`vdFlagCaptureIsForRecord`

`vdFlagCaptureLowLatency`

`vdFlagCaptureAlwaysUseTimeBase`

Return Value
An error return of type `ComponentResult`. See `Error Codes`. Returns `noErr` if there is no error.

Discussion

It has long been a problem for VDIG writers that the sequence grabber can make a series of calls to a VDIG and it is not always clear what their intent is. This function lets you provide additional information about what is happening at the sequence grabber level to the VDIG, so it can take this into account. In particular, the settings bracketing calls are designed for the VDIG to update a series of parameters without reinitializing.

Version Notes

Introduced in QuickTime 6.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`QuickTimeComponents.h`

VDClearClipRgn

Disables all or part of a clipping region that was previously set with VDSetClipRgn.

```
VideoDigitizerError VDClearClipRgn (
   VideoDigitizerComponent ci,
   RgnHandle clipRegion
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

clipRegion

A handle to a `MacRegion` structure that defines the clipping region to clear. This region must correspond to all or part of the clipping region established previously with `VDSetClipRgn` (page 2319).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

VDCompressDone

Determines whether the video digitizer has finished digitizing and compressing a frame of image data.

```
VideoDigitizerError VDCompressDone (
   VideoDigitizerComponent ci,
   UInt8 *queuedFrameCount,
   Ptr *theData,
   long *dataSize,
   UInt8 *similarity,
   TimeRecord *t
);
```

Parameters

ci

Identifies the application's connection to the video digitizer component. An application obtains this value from `OpenComponent` or `OpenDefaultComponent`.

queuedFrameCount

A pointer to the number of queued frames yet to be done. 0 means no frames. Some VDIGs may return 2 even if more than 2 frames are available, and some will return 1 if any number more than 0 are available.

theData

A pointer to a field that is to receive a pointer to the compressed image data. The digitizer returns a pointer that is valid in the application's current memory mode.

dataSize

A pointer to a field to receive a value indicating the number of bytes of compressed image data.

similarity

A pointer to a field to receive an indication of the relative similarity of this image to the previous image in a sequence. A value of 0 indicates that the current frame is a key frame in the sequence. A value of 255 indicates that the current frame is identical to the previous frame. Values from 1 through 254 indicate relative similarity, ranging from very different (1) to very similar (254). This field is only filled in if the temporal quality passed in with `VDSetCompression` (page 2320) is not 0; that is, if it is not frame-differenced.

t

A pointer to a `TimeRecord` structure. When the operation is complete, the digitizer fills in this structure with information indicating when the frame was grabbed. The time value stored in this structure is in the time base that the application sets with `VDSetTimeBase` (page 2337).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

VDCompressOneFrameAsync

Instructs the video digitizer to digitize and compress a single frame of image data.

```
VideoDigitizerError VDCompressOneFrameAsync (
   VideoDigitizerComponent ci
);
```

Parameters

ci

Identifies the application's connection to the video digitizer component. An application obtains this value from `OpenComponent` or `OpenDefaultComponent`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Unlike `VDGrabOneFrameAsync` (page 2308), this function causes the video digitizer to handle all details of managing data buffers.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

VDDone

Determines if VDGrabOneFrameAsync is finished with a specific output buffer.

```
VideoDigitizerError VDDone (
   VideoDigitizerComponent ci,
   short buffer
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

buffer

Identifies the buffer for the operation. The value of this parameter must correspond to a valid index into the list of buffers you supply when your application calls `VDSetupBuffers` (page 2338). This value is zero-based; that is, you must set this parameter to 0 to refer to the first buffer in the buffer list.

Return Value

Returns a long integer indicating whether the specified asynchronous frame grab is complete. If the returned value is 0, the video digitizer component is still working on the frame. If the returned value is nonzero, the digitizer component is finished with the frame and the application can perform its processing.

Discussion

Applications can determine whether a video digitizer component supports asynchronous frame grabbing by examining the output capability flags of the digitizer component, using `VDGetCurrentFlags` (page 2281). Specifically, if the `digiOutDoesAsyncGrabs` flag is set to 1, the digitizer component supports both this function and `VDGrabOneFrameAsync` (page 2308).

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

VDGetActiveSrcRect

Obtains size and location information for the active source rectangle used by a video digitizer component.

```
VideoDigitizerError VDGetActiveSrcRect (
   VideoDigitizerComponent ci,
   short inputStd,
   Rect *activeSrcRect
);
```

Parameters

ci
The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

inputStd
A short integer that specifies the input video signal associated with this maximum source rectangle.

activeSrcRect
A pointer to a `Rect` structure that is to receive the size and location information for the active source rectangle.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Special Considerations
All video digitizer components must support this function.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

VDGetBlackLevelValue

Returns the current black level value.

```
VideoDigitizerError VDGetBlackLevelValue (
   VideoDigitizerComponent ci,
   unsigned short *blackLevel
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

blackLevel

A pointer to an integer field that is to receive the current black level value. Black level values range from 0 to 65,535, where 0 represents the maximum black value and 65,535 represents the minimum black value.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

ExampleVideoPanel

ExampleVideoPanel.win

Declared In

`QuickTimeComponents.h`

VDGetBrightness

Returns the current brightness value.

```
VideoDigitizerError VDGetBrightness (
   VideoDigitizerComponent ci,
   unsigned short *brightness
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

brightness

A pointer to an integer field that is to receive the current brightness value. Brightness values range from 0 to 65,535, where 0 is the darkest possible setting and 65,535 is the lightest possible setting.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

VDGetClipState

Determines whether clipping is enabled.

```
VideoDigitizerError VDGetClipState (
   VideoDigitizerComponent ci,
   short *clipEnable
);
```

Parameters

ci
The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

clipEnable
A pointer to a short integer field that is to receive a value indicating whether clipping is enabled. The video digitizer component places 0 into the field if clipping is disabled, and 1 if it is enabled.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

VDGetCLUTInUse

Obtains the color lookup table used by a video digitizer component.

```
VideoDigitizerError VDGetCLUTInUse (
   VideoDigitizerComponent ci,
   CTabHandle *colorTableHandle
);
```

Parameters

ci
The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

colorTableHandle
A pointer to a field that is to receive a handle to a `ColorTable` structure. The video digitizer component returns a handle to its color lookup table. Applications can then set the destination to use this returned `ColorTable` structure. Your application is responsible for disposing of this handle.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

VDGetCompressionTime

Confirms or quantifies a video digitizer's compression settings.

```
VideoDigitizerError VDGetCompressionTime (
   VideoDigitizerComponent ci,
   OSType compressionType,
   short depth,
   Rect *srcRect,
   CodecQ *spatialQuality,
   CodecQ *temporalQuality,
   unsigned long *compressTime
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

compressionType

A compressor type. This value corresponds to the `component` subtype of the compressor component. See `Codec Identifiers`.

depth

The depth at which the image is to be compressed. Values of 1, 2, 4, 8, 16, 24, and 32 indicate the number of bits per pixel for color images. Values of 33, 34, 36, and 40 indicate 1-bit, 2-bit, 4-bit, and 8-bit grayscale, respectively, for grayscale images.

srcRect

A pointer to a `Rect` structure that defines the portion of the source image to compress.

spatialQuality

A pointer to a field containing the desired compressed image quality (see below). The compressor sets this field to the closest actual quality that it can achieve. A value of `NIL` indicates that the client does not want this information. See these constants:

`codecMinQuality`

`codecLowQuality`

`codecNormalQuality`

`codecHighQuality`

`codecMaxQuality`

`codecLosslessQuality`

temporalQuality

A pointer to a field containing the desired sequence temporal quality (see below). The compressor sets this field to the closest actual quality that it can achieve. A value of `NIL` indicates that the client does not want this information.

compressTime

A pointer to a field to receive the compression time, in milliseconds. Your component should return a long integer indicating the maximum number of milliseconds it would require to compress the specified image. If your component cannot determine the amount of time required to compress the image, set this field to 0. A value of `NIL` indicates that the client does not want this information.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The sequence grabber's video compression settings dialog box uses this function to snap the quality slider to the correct value when working with a compression type that is specified by the video digitizer.

Version Notes

In QuickTime 1.5, video digitizers could provide compressed data directly to clients; however, there was no way to preflight the settings for compression. In QuickTime 2.1, this function was added to allow the video digitizer to quantify the compression time for the actual quality levels that will be used.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

VDGetCompressionTypes

Determines the image-compression capabilities of the video digitizer.

```
VideoDigitizerError VDGetCompressionTypes (
   VideoDigitizerComponent ci,
   VDCompressionListHandle h
);
```

Parameters

ci

Identifies an application's connection to the video digitizer component. An application obtains this value from `OpenComponent` or `OpenDefaultComponent`.

h

A handle to receive the compression information in one or more `VDCompressionList` structures. If the digitizer supports more than one compression type, it creates an array of structures in this handle. The video digitizer returns information about its capabilities by formatting these structures.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Special Considerations

There must be a decompressor component of the appropriate type available in the system if an application is to display images from a compressed image sequence.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

VDGetContrast

Returns the current contrast value.

```
VideoDigitizerError VDGetContrast (
   VideoDigitizerComponent ci,
   unsigned short *contrast
);
```

Parameters

ci
: The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

contrast
: A pointer to an integer field that is to receive the current contrast value. The contrast value ranges from 0 to 65,535, where 0 represents no change to the basic image and larger values increase the contrast of the video image (they increase the slope of the transform).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

VDGetCurrentFlags

Returns status information about a specified video digitizer component.

```
VideoDigitizerError VDGetCurrentFlags (
   VideoDigitizerComponent ci,
   long *inputCurrentFlag,
   long *outputCurrentFlag
);
```

Parameters

ci
: The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

inputCurrentFlag

A pointer to a long integer that is to receive the current input state flags for the video digitizer component; see `Video Digitizer Capabilities`.

outputCurrentFlag

A pointer to a long integer that is to receive the current output state flags for the video digitizer component; see `Video Digitizer Capabilities`.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function is often more convenient than `VDGetDigitizerInfo` (page 2284). For example, this function provides a simple mechanism for determining whether a video digitizer is receiving a valid input signal. An application can retrieve the current input state flags and test the high-order bit by examining the sign of the returned value. If the value is negative (that is, the high-order bit, `digiInSignalLock`, is set to 1), the digitizer component is receiving a valid input signal.

Special Considerations

All video digitizer components must support this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CompressMovies

DigitizerShell

DragAndDrop Shell

Fiendishthngs

MovieGWorlds

Declared In

`QuickTimeComponents.h`

VDGetDataRate

Retrieves information that describes the performance capabilities of a video digitizer.

```
VideoDigitizerError VDGetDataRate (
   VideoDigitizerComponent ci,
   long *milliSecPerFrame,
   Fixed *framesPerSecond,
   long *bytesPerSecond
);
```

Parameters

ci

Identifies the application's connection to the video digitizer component. An application obtains this value from `OpenComponent` or `OpenDefaultComponent`.

milliSecPerFrame
: A pointer to a long integer. The video digitizer returns a value that indicates the number of milliseconds of synchronous overhead involved in digitizing a single frame. This value includes the average delay incurred between the time when the digitizer requests a frame from its associated device, and the time at which the device delivers the frame.

framesPerSecond
: A pointer to a fixed value. The video digitizer supplies the maximum rate at which it can capture video. Note that this value may differ from the rate that the application set with `VDSetFrameRate` (page 2326).

bytesPerSecond
: A pointer to a long integer. Video digitizers that can return compressed image data return a value that indicates the approximate number of bytes per second that the digitizer is generating compressed data, given the current compression and frame rate settings.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

BrideOfMungGrab

Fiendishthngs

Declared In

`QuickTimeComponents.h`

VDGetDeviceNameAndFlags

Returns the current name and device visibility of a video digitizer.

```
VideoDigitizerError VDGetDeviceNameAndFlags (
   VideoDigitizerComponent ci,
   Str255 outName,
   UInt32 *outNameFlags
);
```

Parameters

ci
: The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

outName
: The video digitizer device name.

outNameFlags
: A pointer to a constant (see below) that determines whether to show or hide the VDIG device. See these constants:

    `vdDeviceFlagShowInputsAsDevices`

    `vdDeviceFlagHideDevice`

Return Value
An error return of type `ComponentResult`. See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This routine is designed to give the VDIG more control over how it is presented to the user, and to clarify the distinction between devices and inputs. Historically, the assumption has been that there is one component registered per device and that the component name is displayed. This function lets a component choose its name after registration. When this function is called, it is also a good time to check for hardware and register further VDIG components if needed, allowing for lazy initialization when the application needs to find a VDIG rather than initializing at every launch or replug.

Version Notes
Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
Fiendishthngs

Declared In
`QuickTimeComponents.h`

VDGetDigitizerInfo

Returns capability and status information about a specified video digitizer component.

```
VideoDigitizerError VDGetDigitizerInfo (
   VideoDigitizerComponent ci,
   DigitizerInfo *info
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

info

A pointer to a `DigitizerInfo` structure. The function returns information describing the capabilities of the specified video digitizer into this structure.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Special Considerations

All video digitizer components must support this function.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
DigitizerShell

Declared In
`QuickTimeComponents.h`

VDGetDigitizerRect

Returns the current digitizer rectangle.

```
VideoDigitizerError VDGetDigitizerRect (
   VideoDigitizerComponent ci,
   Rect *digitizerRect
);
```

Parameters

`ci`

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

`digitizerRect`

A pointer to a `Rect` structure that is to receive the size and location information for the current digitizer rectangle.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
All video digitizer components must support this function.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

VDGetDMADepths

Determines which pixel depths a digitizer supports.

```
VideoDigitizerError VDGetDMADepths (
   VideoDigitizerComponent ci,
   long *depthArray,
   long *preferredDepth
);
```

Parameters

`ci`

Identifies the application's connection to the video digitizer component. An application obtains this value from `OpenComponent` or `OpenDefaultComponent`.

`depthArray`

A pointer to a long integer. The video digitizer returns a value that indicates the depths it can support. Each depth is represented by a single bit in this field. More than one bit may be set to 1.

preferredDepth

A pointer to a long integer. Video digitizers that have a preferred depth value return that value in this field, using one of the possible values of the `depthArray` parameter. Digitizers that do not prefer any given value set this field to 0.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The flags returned by this function augment the information that an application can obtain from the digitizer's output capability flags in the `DigitizerInfo` structure. If a digitizer does not support this function but does support DMA, an application may assume that the digitizer can handle offscreen buffers at all of the depths indicated in its output capabilities flags. Applications may use the following enumerators to set bits in the field referred to by the `depthArray` parameter.

```
enum {
    dmaDepth1        =1     /* supports black and white */
    dmaDepth2        =2     /* supports 2-bit color */
    dmaDepth4        =4     /* supports 4-bit color */
    dmaDepth8        =8     /* supports 8-bit color */
    dmaDepth16       =16    /* supports 16-bit color */
    dmaDepth32       =32    /* supports 32-bit color */
    dmaDepth2Gray    =64    /* supports 2-bit grayscale */
    dmaDepth4Gray    =128   /* supports 4-bit grayscale */
    dmaDepth8Gray    =256   /* supports 8-bit grayscale */
};
```

Special Considerations

Before a program that uses a video digitizer creates an offscreen buffer, it should call the this function to determine the pixel depths supported by the digitizer. If possible, the program should use the preferred depth, in order to obtain the best possible display performance.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

VDGetFieldPreference

Determines which field is being used in cases where the image is vertically scaled to half its original size.

```
VideoDigitizerError VDGetFieldPreference (
   VideoDigitizerComponent ci,
   short *fieldFlag
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

fieldFlag

Points to a field that is to receive a value (see below) indicating which field is being used. See these constants:

`vdUseAnyField`

`vdUseOddField`

`vdUseEvenField`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

VDGetHue

Returns the current hue value.

```
VideoDigitizerError VDGetHue (
   VideoDigitizerComponent ci,
   unsigned short *hue
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

hue

A pointer to an integer that is to receive the current hue value. Hue is similar to the tint control on a television, and it is specified in degrees with complementary colors set 180 degrees apart (red is 0 degrees, green is +120 degrees, and blue is -120 degrees). Video digitizer components support hue values that range from 0 (-180 degrees shift in hue) to 65,535 (+179 degrees shift in hue), where 32,767 represents a 0 degree shift in hue.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

VDGetImageDescription

Retrieves an ImageDescription structure from a video digitizer.

```
VideoDigitizerError VDGetImageDescription (
   VideoDigitizerComponent ci,
   ImageDescriptionHandle desc
);
```

Parameters

ci

Identifies the application's connection to the video digitizer component. An application obtains this value from `OpenComponent` or `OpenDefaultComponent`.

desc

A handle. The video digitizer fills this handle with an `ImageDescription` structure containing information about the digitizer's current compression settings. The digitizer resizes the handle appropriately. It is the application's responsibility to dispose of this handle.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Fiendishthngs

Declared In

`QuickTimeComponents.h`

VDGetInput

Returns data that identifies the currently active input video source.

```
VideoDigitizerError VDGetInput (
   VideoDigitizerComponent ci,
   short *input
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

input

A pointer to a short integer that is to receive the identifier for the currently active input video source. Video digitizer components number video sources sequentially, starting at 0. So, if the first source is active, this function sets the field referred to by the `input` parameter to 0.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Special Considerations

All video digitizer components must support this function.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Fiendishthngs

Declared In
`QuickTimeComponents.h`

VDGetInputColorSpaceMode

Determines whether a digitizer is operating in color or grayscale mode.

```
VideoDigitizerError VDGetInputColorSpaceMode (
   VideoDigitizerComponent ci,
   short *colorSpaceMode
);
```

Parameters

`ci`

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

`colorSpaceMode`

A pointer to a value that indicates whether the digitizer is operating in color (1) or grayscale (0) mode.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Applications can determine whether a digitizer component supports grayscale or color digitization by examining the digitizer component's input capability flags. Specifically, if the `digiInDoesColor` flag is set to 1, the digitizer component supports color digitization. Similarly, if the `digiInDoesBW` flag is set to 1, the digitizer component supports grayscale digitization. Applications can use `VDGetCurrentFlags` (page 2281) to obtain the input capability flags of a digitizer component.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

VDGetInputFormat

Determines the format of the video signal provided by a specified video input source.

```
VideoDigitizerError VDGetInputFormat (
   VideoDigitizerComponent ci,
   short input,
   short *format
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

input

The input video source for this request. Video digitizer components number video sources sequentially, starting at 0. So, to request information about the first video source, an application sets this parameter to 0. Applications can get the number of video sources supported by a video digitizer component by calling `VDGetNumberOfInputs` (page 2297).

format

A pointer to a short integer that is to receive a constant (see below) that specifies the video format of the specified input source. See these constants:

`compositeIn`

`sVideoIn`

`rgbComponentIn`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

All video digitizer components must support this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

VDGetInputGammaRecord

Retrieves a pointer to the active input VDGammaRecord structure for a video digitizer.

```
VideoDigitizerError VDGetInputGammaRecord (
   VideoDigitizerComponent ci,
   VDGamRecPtr *inputGammaPtr
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

inputGammaPtr

A pointer to a field that is to receive a pointer to an input `VDGammaRecord` structure.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Gamma structures give applications complete control over color filtering transforms and are therefore more precise than the gamma values that can be set by calling `VDSetInputGammaValue` (page 2329).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

VDGetInputGammaValue

Returns the current gamma values.

```
VideoDigitizerError VDGetInputGammaValue (
   VideoDigitizerComponent ci,
   Fixed *channel1,
   Fixed *channel2,
   Fixed *channel3
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

channel1

A pointer to a fixed integer field that is to receive the gamma value for the red component of the input video signal.

channel2

A pointer to a fixed integer field that is to receive the gamma value for the green component of the input video signal.

channel3

A pointer to a fixed integer field that is to receive the gamma value for the blue component of the input video signal.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

VDGetInputName

Gets the name of a video input.

```
VideoDigitizerError VDGetInputName (
   VideoDigitizerComponent ci,
   long videoInput,
   Str255 name
);
```

Parameters

ci

Specifies the video digitizer component for this operation. Applications can obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

videoInput

The input video source for this request. Video digitizer components number video sources sequentially, starting at 0. So, to request information about the first video source, an application sets this parameter to 0. Applications can get the number of video sources supported by a video digitizer component by calling `VDGetNumberOfInputs` (page 2297).

name

The video input source's name string.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

Fiendishthngs

Declared In

`QuickTimeComponents.h`

VDGetKeyColor

Obtains the index value of the active key color.

```
VideoDigitizerError VDGetKeyColor (
   VideoDigitizerComponent ci,
   long *index
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

index

A pointer to a field that is to receive the index of the key color. This index value identifies the key color within the currently active color lookup table. If there are several active key colors, the video digitizer returns the first color from the key color list. Subsequently, applications use `VDGetNextKeyColor` (page 2296) to obtain other colors from the list. If there is no active key color, the function sets the field to -1.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Special Considerations

All video digitizer components that support key colors must support this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

VDGetKeyColorRange

Obtains the currently defined key color range.

```
VideoDigitizerError VDGetKeyColorRange (
   VideoDigitizerComponent ci,
   RGBColor *minRGB,
   RGBColor *maxRGB
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

minRGB

A pointer to a field that is to receive the lower bound of the key color range. The video digitizer component places the `RGBColor` structure that corresponds to the lower end of the range in the field referred to by this parameter.

maxRGB

A pointer to a field that is to receive the upper bound of the key color range. The video digitizer component places the `RGBColor` structure that corresponds to the upper end of the range in the field referred to by this parameter.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

VDGetMaskandValue

Obtains the appropriate alpha channel or blend mask value for a desired level of video blending.

```
VideoDigitizerError VDGetMaskandValue (
   VideoDigitizerComponent ci,
   unsigned short blendLevel,
   long *mask,
   long *value
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

blendLevel

The desired blend level. Valid values range from 0 to 65,535, where 0 corresponds to no video and 65,535 corresponds to all video.

mask

A pointer to a field that is to receive a value indicating which bits are meaningful in the data returned for the `value` parameter. The video digitizer component sets to 1 the bits that correspond to meaningful bits in the data returned for the `value` parameter.

value

A pointer to a field that is to receive data that can be used to obtain the desired blend level. The data returned for the `mask` parameter indicates which bits are valid in the data returned for this parameter.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
The information returned by the digitizer component differs based on the type of blending supported by the component. In all cases, however, the returned value of the `value` parameter contains the value for the desired blend level, and the returned value of the `mask` parameter indicates which bits in the `value` parameter are meaningful. Bits in the returned mask parameter value that are set to 1 correspond to meaningful bits in the returned value parameter value.

For example, if an application requests a 50 percent video blend level from a digitizer that supports 8-bit alpha channels, the digitizer component might return 0xFF000000 in the `mask` parameter, identifying a full upper byte as the alpha channel, and 0x80000000 in the `value` parameter, specifying a 50 percent blend level.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

VDGetMaskPixMap

Retrieves the pixel map data for a component's blend mask.

```
VideoDigitizerError VDGetMaskPixMap (
   VideoDigitizerComponent ci,
   PixMapHandle maskPixMap
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

maskPixMap

A handle to a `PixMap` structure. The video digitizer component returns the pixel map data for its blend mask into the `PixMap` structure specified by this parameter. The video digitizer component resizes the handle as appropriate. Your application is responsible for disposing of this handle.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function is supported only by digitizer components that support blend masks.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

VDGetMaxAuxBuffer

Obtains access to buffers that are located on special hardware.

```
VideoDigitizerError VDGetMaxAuxBuffer (
   VideoDigitizerComponent ci,
   PixMapHandle *pm,
   Rect *r
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

pm

A pointer to a handle to a `PixMap` structure. The video digitizer component returns a handle to the destination `PixMap` structure in the field referred to by this parameter. Do not dispose of this structure. If the digitizer component cannot allocate a buffer, this handle is set to `NIL`.

r

A pointer to a `Rect` structure. The video digitizer component places the coordinates of the largest output rectangle it can support into the structure referred to by this parameter.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

VDGetMaxSrcRect

Returns the maximum source rectangle.

```
VideoDigitizerError VDGetMaxSrcRect (
   VideoDigitizerComponent ci,
   short inputStd,
   Rect *maxSrcRect
);
```

Parameters

ci
The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

inputStd
A short integer that specifies the input video signal associated with this maximum source rectangle.

maxSrcRect
A pointer to a `Rect` structure that is to receive the size and location information for the maximum source rectangle.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
All video digitizer components must support this function.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

VDGetNextKeyColor

Obtains the index value of the active key colors in cases where the digitizer component supports multiple key colors.

```
VideoDigitizerError VDGetNextKeyColor (
   VideoDigitizerComponent ci,
   long index
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

index

A field that is to receive the index of the next key color. This index value identifies the key color within the currently active color lookup table. If there are no more colors left in the list, the digitizer component sets the field referred to by the `index` parameter to -1.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

All video digitizer components that support multiple key colors must support this function

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

VDGetNumberOfInputs

Returns the number of input video sources that a video digitizer component supports.

```
VideoDigitizerError VDGetNumberOfInputs (
   VideoDigitizerComponent ci,
   short *inputs
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

inputs

A pointer to an integer that is to receive the number of input video sources supported by the specified component. Video digitizer components number video sources sequentially, starting at 0. So, if a digitizer component supports two inputs, this function sets the field referred to by the `inputs` parameter to 1.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

All video digitizer components must support this function.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Fiendishthngs

Declared In
`QuickTimeComponents.h`

VDGetPlayThruDestination

Obtains information about the current video destination.

```
VideoDigitizerError VDGetPlayThruDestination (
   VideoDigitizerComponent ci,
   PixMapHandle *dest,
   Rect *destRect,
   MatrixRecord *m,
   RgnHandle *mask
);
```

Parameters

ci
: The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

dest
: A pointer to a handle to a `PixMap` structure. The video digitizer component returns a handle to the destination `PixMap` structure in the field referred to by this parameter. It is the caller's responsibility to dispose of the `PixMap` structure.

destRect
: A pointer to a `Rect` structure. The video digitizer component places the coordinates of the output rectangle into the structure referred to by this parameter. If there is no output rectangle defined, the component returns an empty rectangle.

m
: A pointer to a `MatrixRecord` structure. The video digitizer component places the transformation matrix into the structure referred to by this parameter.

mask
: A pointer to a handle to a `MacRegion` structure. The video digitizer component places a handle to the mask region into the field referred to by this parameter. Applications can use masks to control the video into the destination rectangle. If there is no mask region defined, the digitizer component sets this returned handle to `NIL`. The caller is responsible for disposing of the `MacRegion` structure.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
All video digitizer components must support this function.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

VDGetPLLFilterType

Determines which phase locked loop (PLL) mode is currently active for a video digitizer.

```
VideoDigitizerError VDGetPLLFilterType (
   VideoDigitizerComponent ci,
   short *pllType
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

pllType

Points to a field that is to receive a value indicating which PLL mode is active. Values are 0 for broadcast mode and 1 for videotape recorder mode.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

VDGetPreferredImageDimensions

Gets the preferred image dimensions for a video digitizer.

```
VideoDigitizerError VDGetPreferredImageDimensions (
   VideoDigitizerComponent ci,
   long *width,
   long *height
);
```

Parameters

ci

Specifies the video digitizer component for this operation. Applications can obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

width

A pointer to the preferred image width.

height

A pointer to the preferred image height.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

VDGetPreferredTimeScale

Determines a digitizer's preferred time scale.

```
VideoDigitizerError VDGetPreferredTimeScale (
   VideoDigitizerComponent ci,
   TimeScale *preferred
);
```

Parameters

ci

Identifies the application's connection to the video digitizer component. An application obtains this value from `OpenComponent` or `OpenDefaultComponent`.

preferred

A pointer to a time scale. The video digitizer returns information about its preferred time scale in this structure.

Return Value

If the digitizer does not have a preferred time scale, it returns a result code of `digiUnimpErr`. See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

VDGetSaturation

Returns the current saturation value.

```
VideoDigitizerError VDGetSaturation (
   VideoDigitizerComponent ci,
   unsigned short *saturation
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

saturation

A pointer to an integer that is to receive the current saturation value. The saturation value controls color intensity. For example, at high saturation levels, red appears to be red; at low saturation, red appears as pink. Valid saturation values range from 0 to 65,535, where 0 is the minimum saturation value and 65,535 specifies maximum saturation.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

VDGetSharpness

Returns the current sharpness value.

```
VideoDigitizerError VDGetSharpness (
   VideoDigitizerComponent ci,
   unsigned short *sharpness
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

sharpness

A pointer to an integer that is to receive the current sharpness value. The sharpness value ranges from 0 to 65,535, where 0 represents no sharpness filtering and 65,535 represents full sharpness filtering. Higher values result in a visual impression of increased picture sharpness.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

VDGetSoundInputDriver

Retrieves information about a video digitizer's sound input driver.

```
VideoDigitizerError VDGetSoundInputDriver (
   VideoDigitizerComponent ci,
   Str255 soundDriverName
);
```

Parameters

ci

Identifies the application's connection to the video digitizer component. An application obtains this value from `OpenComponent` or `OpenDefaultComponent`.

soundDriverName

A pointer to a string. The video digitizer returns the name of its sound input driver. If the digitizer does not have an associated driver, it returns a result code of `digiUnimpErr`.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Fiendishthngs

Declared In
`QuickTimeComponents.h`

VDGetSoundInputSource

Instructs your video digitizer component to return the sound input source associated with a particular video input.

```
VideoDigitizerError VDGetSoundInputSource (
   VideoDigitizerComponent ci,
   long videoInput,
   long *soundInput
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

videoInput
: The input video source for this request. Video digitizer components number video sources sequentially, starting at 0. So, to request information about the first video source, an application sets this parameter to 0. Applications can get the number of video sources supported by a video digitizer component by calling `VDGetNumberOfInputs` (page 2297).

soundInput
: The sound input index to use with the sound input driver returned by `VDGetSoundInputDriver` (page 2302).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

Some video digitizers may associate different sound inputs with each video input. `VDGetSoundInputDriver` (page 2302) returns the name of the sound input driver that the sound input is associated with.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

VDGetTimeCode

Instructs your video digitizer component to return timecode information for the incoming video signal.

```
VideoDigitizerError VDGetTimeCode (
   VideoDigitizerComponent ci,
   TimeRecord *atTime,
   void *timeCodeFormat,
   void *timeCodeTime
);
```

Parameters

ci
: The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

atTime
: A pointer to a `TimeRecord` structure to receive the QuickTime movie time value corresponding to the timecode information.

timeCodeFormat
: A pointer to a `TimeCodeDef` structure. Your video digitizer component returns the movie's timecode definition information in this structure.

timeCodeTime
: A pointer to a `TimeCodeRecord` structure. Your video digitizer component returns the time value corresponding to the movie time contained in this structure.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Typically, this function is called once, at the beginning of a capture session. The use of this function assumes that the timecoding for the entire capture session will be continuous.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

VDGetUniqueIDs

Returns a unique identifier for a particular video digitizer device.

```
VideoDigitizerError VDGetUniqueIDs (
   VideoDigitizerComponent ci,
   UInt64 *outDeviceID,
   UInt64 *outInputID
);
```

Parameters

ci
The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

outDeviceID
A pointer to a 64-bit hardware device ID. In the case of a FireWire device, this is the FireWire ID.

outInputID
A pointer to a 64-bit hardware input ID. A return of 0 means you don't have one.

Return Value
An error return of type `ComponentResult`. See `Error Codes`. Returns `noErr` if there is no error.

Discussion
This function is provided so the VDIG can give the sequence grabber information that helps it choose a particular device and input from those available. You might use it, for example, to restore a specific camera from a set of several hot-plugged FireWire cameras. The caller can pass `NIL` if it is not interested in one of the IDs.

Version Notes
Introduced in QuickTime 6.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`QuickTimeComponents.h`

VDGetVBlankRect

Returns the vertical blanking rectangle.

```
VideoDigitizerError VDGetVBlankRect (
   VideoDigitizerComponent ci,
   short inputStd,
   Rect *vBlankRect
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

inputStd

A short integer (see below) that identifies the signaling standard used in the source video signal. See these constants:

`ntscIn`

`palIn`

`secamIn`

vBlankRect

A pointer to a `Rect` structure that is to receive the size and location information for the vertical blanking rectangle.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

All video digitizer components must support this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

VDGetVideoDefaults

Returns the recommended values for many of the analog video parameters that may be set by applications.

```
VideoDigitizerError VDGetVideoDefaults (
   VideoDigitizerComponent ci,
   unsigned short *blackLevel,
   unsigned short *whiteLevel,
   unsigned short *brightness,
   unsigned short *hue,
   unsigned short *saturation,
   unsigned short *contrast,
   unsigned short *sharpness
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

blackLevel

A pointer to an integer that is to receive the default black level value. Black level values range from 0 to 65,535, where 0 represents the maximum black value and 65,535 represents the minimum black value.

whiteLevel

A pointer to an integer that is to receive the default white level value. White level values range from 0 to 65,535, where 0 represents the minimum white value and 65,535 represents the maximum white value.

brightness

A pointer to an integer that is to receive the default brightness value. Brightness values range from 0 to 65,535, where 0 is the darkest possible setting and 65,535 is the lightest possible setting.

hue

A pointer to an integer that is to receive the default hue value. Hue is similar to the tint control on a television, and it is specified in degrees with complementary colors set 180 degrees apart (red is 0 degrees, green is +120 degrees, and blue is -120 degrees). Video digitizer components support hue values that range from 0 (-180 degrees shift in hue) to 65,535 (+179 degrees shift in hue), where 32,767 represents a 0 degree shift in hue.

saturation

A pointer to an integer that is to receive the default saturation value. The saturation value controls color intensity. For example, at high saturation levels, red appears to be red; at low saturation, red appears as pink. Valid saturation values range from 0 to 65,535, where 0 is the minimum saturation value and 65,535 specifies maximum saturation.

contrast

A pointer to an integer that is to receive the default contrast value. The contrast value ranges from 0 to 65,535, where 0 represents no change to the basic image and larger values increase the contrast of the video image (they increase the slope of the transform).

sharpness

A pointer to an integer that is to receive the default sharpness value. The sharpness value ranges from 0 to 65,535, where 0 represents no sharpness filtering and 65,535 represents full sharpness filtering. Higher values result in a visual impression of increased picture sharpness.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

All video digitizer components must support this function.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

VDGetWhiteLevelValue

Returns the current white level value.

```
VideoDigitizerError VDGetWhiteLevelValue (
   VideoDigitizerComponent ci,
   unsigned short *whiteLevel
);
```

Parameters

`ci`
: The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

`whiteLevel`
: A pointer to an integer that is to receive the current white level value. White level values range from 0 to 65,535, where 0 represents the minimum white value and 65,535 represents the maximum white value.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

VDGrabOneFrame

Instructs the video digitizer component to digitize a single frame of source video.

```
VideoDigitizerError VDGrabOneFrame (
   VideoDigitizerComponent ci
);
```

Parameters

`ci`
: The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion

If the specified digitizer component is already digitizing continuously when the application calls this function, the digitizer component returns the next digitized frame and then stops. If the digitizer component is stopped, the component digitizes a single frame and then stops. To resume continuous digitization, applications should call `VDSetPlayThruOnOff` (page 2334).

Special Considerations

This function supports synchronous single-frame video digitization; that is, the digitizer component does not return control to your application until it has successfully processed the next video frame. Some video digitizer components may also support asynchronous single-frame digitization. Applications can request asynchronous digitization by calling `VDGrabOneFrameAsync` (page 2308).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

VDGrabOneFrameAsync

Instructs the video digitizer component to start to digitize asynchronously a single frame of source video.

```
VideoDigitizerError VDGrabOneFrameAsync (
   VideoDigitizerComponent ci,
   short buffer
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

buffer

Identifies the next output buffer. The value of this parameter must correspond to a valid index into the list of buffers that you supply when your application calls `VDSetupBuffers` (page 2338). Note that this value is zero-based (that is, you must set this parameter to 0 to refer to the first buffer in the buffer list).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

When calling this function, the application specifies the next destination video buffer, allowing the digitizer component to quickly switch from the current buffer to the next buffer. In this manner, your application's ability to grab video at high frame rates is enhanced. If the specified digitizer component is already digitizing continuously when the application calls this function, the digitizer component returns the next digitized frame and then stops. If the digitizer component is stopped, the component digitizes a single frame and then stops. To resume continuous digitization, applications should call `VDSetPlayThruOnOff` (page 2334).

This function also allows applications to use more than one destination buffer for the digitized video. The application defines these buffers by calling `VDSetupBuffers` (page 2338). The application specifies one of these destination buffers for the digitized frame when it calls `VDSetPlayThruDestination` (page 2332) or `VDSetPlayThruGlobalRect` (page 2333).

Special Considerations

Applications can determine whether a video digitizer component supports asynchronous frame grabbing by using `VDGetCurrentFlags` (page 2281) to retrieve the digitizer component's output capability flags. If the `digiOutDoesAsyncGrabs` flag is set to 1, the digitizer component supports both this function and `VDDone` (page 2275). If a video digitizer component does not support asynchronous digitization, applications must use `VDGrabOneFrame` (page 2307) to perform single-frame digitization.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

VDIIDCGetCSRData

Reads a camera's CSR registers directly.

```
VideoDigitizerError VDIIDCGetCSRData (
   VideoDigitizerComponent ci,
   Boolean offsetFromUnitBase,
   UInt32 offset,
   UInt32 *data
);
```

Parameters

`ci`

The component instance that identifies your connection to a video digitizer component. The digitizer's subtype must be `vdSubtypeIIDC` (`'iidc'`).

`offsetFromUnitBase`

Pass TRUE if the offset is relative to the initial unit space (FFFF Fxxx xxxx), FALSE if the offset is relative to the initial register space (FFFF F000 0000).

`offset`

Offset in bytes of the value to read.

`data`

Location to store the `value` (of type UInt32) that was read.

Return Value

See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error.

Discussion

You might want to read a camera's registers directly if you're querying the state of a feature not accessed by `VDIIDCGetFeatures` or if some camera-specific information must be accessed.

Version Notes

Introduced in QuickTime 6.4.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`QuickTimeComponents.h`

VDIIDCGetDefaultFeatures

Places atoms in a QuickTime atom container that specify the default capabilities and default state of a camera's IIDC features.

```
VideoDigitizerError VDIIDCGetDefaultFeatures (
   VideoDigitizerComponent ci,
   QTAtomContainer *container
);
```

Parameters

ci
: The component instance that identifies your connection to a video digitizer component. The digitizer's subtype must be `vdSubtypeIIDC` (`'iidc'`).

container
: Upon return, a pointer to a QuickTime atom container containing atoms of type `vdIIDCAtomTypeFeature` for each IIDC camera feature whose default is known. The container may be empty if defaults cannot be determined.

Return Value
See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error.

Discussion
The digitizer will create the QuickTime atom container, and it is the responsibility of the client to delete it if the routine does not return an error.

Version Notes
Introduced in QuickTime 6.4.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`QuickTimeComponents.h`

VDIIDCGetFeatures

Places atoms in a QuickTime atom container that specify the current capabilities of a camera and the state of its IIDC features.

```
VideoDigitizerError VDIIDCGetFeatures (
   VideoDigitizerComponent ci,
   QTAtomContainer *container
);
```

Parameters

ci

The component instance that identifies your connection to a video digitizer component. The digitizer's subtype must be `vdSubtypeIIDC` (`'iidc'`).

container

Upon return, a pointer to a QuickTime atom container containing atoms of type `vdIIDCAtomTypeFeature` for each IIDC camera feature. If the camera has not implemented any IIDC features the container returns empty.

Return Value
See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error.

Discussion
The digitizer creates the container, and it is the responsibility of the client to ultimately delete it if the routine does not return an error. Since the values that this function retrieves might change underneath the client, they should not be cached but should be retrieved each time they are needed.

Version Notes
Introduced in QuickTime 6.4.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
Fiendishthngs

Declared In
`QuickTimeComponents.h`

VDIIDCGetFeaturesForSpecifier

Places atoms in a QuickTime atom container that specify the current state of a single camera IIDC feature or group of features.

```
VideoDigitizerError VDIIDCGetFeaturesForSpecifier (
   VideoDigitizerComponent ci,
   OSType specifier,
   QTAtomContainer *container
);
```

Parameters

ci

The component instance that identifies your connection to a video digitizer component. The digitizer's subtype must be `vdSubtypeIIDC` (`'iidc'`).

specifier

The feature or group of features to be retrieved: // IIDC feature types `vdIIDCFeatureHue = 'hue '`, `vdIIDCFeatureSaturation = 'satu'`, `vdIIDCFeatureSharpness = 'shrp'`, `vdIIDCFeatureBrightness = 'brit'`, `vdIIDCFeatureGain = 'gain'`, `vdIIDCFeatureIris = 'iris'`, `vdIIDCFeatureShutter = 'shtr'`, `vdIIDCFeatureExposure = 'xpsr'`, `vdIIDCFeatureWhiteBalanceU = 'whbu'`, `vdIIDCFeatureWhiteBalanceV = 'whbv'`, `vdIIDCFeatureGamma = 'gmma'`, `vdIIDCFeatureTemperature = 'temp'`, `vdIIDCFeatureZoom = 'zoom'`, `vdIIDCFeatureFocus = 'fcus'`, `vdIIDCFeaturePan = 'pan '`, `vdIIDCFeatureTilt = 'tilt'`, `vdIIDCFeatureOpticalFilter = 'opft'`, `vdIIDCFeatureTrigger = 'trgr'`, `vdIIDCFeatureCaptureSize = 'cpsz'`, `vdIIDCFeatureCaptureQuality = 'cpql'`, `vdIIDCFeatureFocusPoint = 'fpnt'`, `vdIIDCFeatureEdgeEnhancement = 'eden'` `vdIIDCFeatureLightingHint = 'lhnt'` // IIDC group types `vdIIDCGroupImage = 'imag'`, `vdIIDCGroupColor = 'colr'`, `vdIIDCGroupMechanics = 'mech'`, `vdIIDCGroupTrigger = 'trig'`

container

Upon return, a pointer to a QuickTime atom container containing atoms of type `vdIIDCAtomTypeFeature` for each IIDC camera feature corresponding to the specifier. If the camera has not implemented any of the specified features the container returns empty.

Return Value

See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error.

Discussion

The digitizer creates the container, and it is the responsibility of the client to ultimately delete it if the routine does not return an error. Since the values that this function retrieves might change underneath the client, they should not be cached but should be retrieved each time they are needed.

Version Notes

Introduced in QuickTime 6.4.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`QuickTimeComponents.h`

VDIIDCSetCSRData

Writes to a camera's CSR registers directly.

```
VideoDigitizerError VDIIDCSetCSRData (
   VideoDigitizerComponent ci,
   Boolean offsetFromUnitBase,
   UInt32 offset,
   UInt32 data
);
```

Parameters

ci

The component instance that identifies your connection to a video digitizer component. The digitizer's subtype must be `vdSubtypeIIDC` (`'iidc'`).

offsetFromUnitBase
: Pass TRUE if the offset is relative to the initial unit space (FFFF Fxxx xxxx), FALSE if the offset is relative to the initial register space (FFFF F000 0000).

offset
: Offset in bytes of the value to set.

data
: Location of the `value` (of type UInt32) to write.

Return Value
See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error.

Discussion
You might want to write to a camera's registers directly if you're setting the state of a feature not accessed by `VDIIDCSetFeatures` or if some camera-specific information must be set.

Version Notes
Introduced in QuickTime 6.4.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`QuickTimeComponents.h`

VDIIDCSetFeatures

Changes the state of a camera's IIDC features.

```
VideoDigitizerError VDIIDCSetFeatures (
   VideoDigitizerComponent ci,
   QTAtomContainer container
);
```

Parameters

ci
: The component instance that identifies your connection to a video digitizer component. The digitizer's subtype must be `vdSubtypeIIDC` (`'iidc'`).

container
: A pointer to a QuickTime atom container populated with atoms of type `vdIIDCAtomTypeFeature`; the container may have one or many atoms in it. An empty container will cause the function to have no effect.

Return Value
See `Error Codes` in the QuickTime API Reference. Returns `noErr` if there is no error.

Discussion
It is the responsibility of the client to provide the QuickTime atom container and delete it after use.

Version Notes
Introduced in QuickTime 6.4.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`QuickTimeComponents.h`

VDPreflightDestination

Verifies that a video digitizer component can support a set of destination settings intended for use with VDSetPlayThruDestination.

```
VideoDigitizerError VDPreflightDestination (
   VideoDigitizerComponent ci,
   Rect *digitizerRect,
   PixMap **dest,
   RectPtr destRect,
   MatrixRecordPtr m
);
```

Parameters

`ci`

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

`digitizerRect`

A pointer to a `Rect` structure that contains the size and location information for the digitizer rectangle. The coordinates of this rectangle must be relative to the maximum source rectangle. In addition, the digitizer rectangle must be within the maximum source rectangle. For a discussion of the relationship between these rectangles, see "Video Digitizer Components" in *Inside Macintosh: QuickTime* Components. If the video digitizer component cannot accommodate the specified rectangle, it changes the coordinates in this structure to specify a rectangle that it can support and sets the result to `qtParamErr`.

`dest`

A handle to the destination `PixMap` structure.

`destRect`

A pointer to a `Rect` structure that specifies the size and location of the video destination. This is an optional parameter. Applications may specify a transformation matrix to control the placement and scaling of the video image in the destination `PixMap` structure. In this case, the `destRect` parameter is set to `NIL` and the `m` parameter specifies the matrix. The destination rectangle must be in the coordinate system of the destination `PixMap` structure specified by the `dest` parameter. If the video digitizer component cannot accommodate this rectangle, it changes the coordinates in the structure to specify a rectangle that it can support and sets the result to `qtParamErr`.

`m`

A pointer to a `MatrixRecord` structure containing the transformation matrix for the destination video image. This is an optional parameter. Applications may specify a destination rectangle to control the placement and scaling of the video image in the destination `PixMap` structure. In this case, the `m` parameter is set to `NIL` and the `destRect` parameter specifies the destination rectangle. If the `destRect` parameter is `NIL`, you can determine the destination rectangle for simple matrices by calling `TransformRect` (page 736) using the current digitizer rectangle and this matrix. If the video digitizer component cannot accommodate this matrix, it changes the values in the structure to define a matrix that it can support and sets the result to `qtParamErr`. Applications can determine the capabilities of a video digitizer component by calling `VDGetDigitizerInfo` (page 2284).

Return Value
The application provides the desired settings as parameters to this function. The video digitizer component then examines those settings. If the digitizer component can support the specified settings, it sets the result code to `noErr`. If the digitizer component cannot support the settings, it alters the input settings to reflect values that it can support and returns a result code of `qtParamErr`. See `Error Codes`.

Discussion
All video digitizer components must support this function. Applications should use this function to test destination settings whenever a video digitizer component cannot support arbitrary scaling.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

VDPreflightGlobalRect

Verifies that a video digitizer component can support a set of destination settings intended for use with VDSetPlayThruGlobalRect.

```
VideoDigitizerError VDPreflightGlobalRect (
   VideoDigitizerComponent ci,
   GrafPtr theWindow,
   Rect *globalRect
);
```

Parameters

`ci`
: The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

`theWindow`
: A pointer to the destination window.

`globalRect`
: A pointer to a `Rect` structure that specifies the size and location of the video destination. This rectangle must be in the coordinate system of the destination window specified by the `theWindow` parameter. If the video digitizer component cannot accommodate this rectangle, it changes the coordinates in the structure to specify a rectangle that it can support and sets the result to `qtParamErr`.

Return Value
Returns `qtParamErr` if the video digitizer component cannot accommodate the destination rectangle. Returns `digiUnimpErr` if the video digitizer component does not support placing destination video into a rectangle that crosses screens. See `Error Codes`. Returns `noErr` if there is no error.

Discussion
Applications should use this function to determine whether a video digitizer supports placing destination video into a rectangle that crosses screens. Digitizers that do not support this capability return a result of `digiUnimpErr`.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

VDReleaseAsyncBuffers

Releases the buffers that were allocated with VDSetupBuffers.

```
VideoDigitizerError VDReleaseAsyncBuffers (
   VideoDigitizerComponent ci
);
```

Parameters

ci
: The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

VDReleaseCompressBuffer

Frees a buffer received from VDCompressDone.

```
VideoDigitizerError VDReleaseCompressBuffer (
   VideoDigitizerComponent ci,
   Ptr bufferAddr
);
```

Parameters

ci
: Identifies the application's connection to the video digitizer component. An application obtains this value from `OpenComponent` or `OpenDefaultComponent`.

bufferAddr
: Points to the location of the buffer to be released. This address must correspond to a buffer address that the application obtained from `VDCompressDone` (page 2273).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

VDResetCompressSequence

Forces the video digitizer to insert a key frame into a temporally compressed image sequence.

```
VideoDigitizerError VDResetCompressSequence (
   VideoDigitizerComponent ci
);
```

Parameters

ci

Identifies the application's connection to the video digitizer component. An application obtains this value from `OpenComponent` or `OpenDefaultComponent`.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

VDSelectUniqueIDs

Selects a video digitizer device by ID.

```
VideoDigitizerError VDSelectUniqueIDs (
   VideoDigitizerComponent ci,
   const UInt64 *inDeviceID,
   const UInt64 *inInputID
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

inDeviceID

A pointer to a unique 64-bit hardware device ID.

inInputID

A pointer to a unique 64-bit hardware input ID.

Return Value
An error return of type `ComponentResult`. See `Error Codes`. Returns `vdDontHaveThatUniqueIDErr` if your device doesn't have a match. Returns `noErr` if there is no error.

Discussion

Note this function does selection, not setting. The assumption is that the unique ID is set by the hardware and is not modifiable by the calling application. Passing either a `NIL` pointer or 0 for an ID means you don't care. This should restore the device and input IDs returned by `VDGetUniqueIDs` (page 2304).

Version Notes

Introduced in QuickTime 6.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`QuickTimeComponents.h`

VDSetBlackLevelValue

Sets the current black level value.

```
VideoDigitizerError VDSetBlackLevelValue (
   VideoDigitizerComponent ci,
   unsigned short *blackLevel
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

blackLevel

A pointer to an integer that contains the new black level value. Black level values range from 0 to 65,535, where 0 represents the maximum black value and 65,535 represents the minimum black value. The digitizer component returns the new value, so that the application can avoid using unsupported values in future requests.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

ExampleVideoPanel

ExampleVideoPanel.win

Declared In

`QuickTimeComponents.h`

VDSetBrightness

Sets the current brightness value.

```
VideoDigitizerError VDSetBrightness (
   VideoDigitizerComponent ci,
   unsigned short *brightness
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

brightness

A pointer to an integer that contains the new brightness value. Brightness values range from 0 to 65,535, where 0 is the darkest possible setting and 65,535 is the lightest possible setting. The digitizer component returns the new value, so that the application can avoid using unsupported values in future requests.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

VDSetClipRgn

Defines a clipping region for a video digitizer.

```
VideoDigitizerError VDSetClipRgn (
   VideoDigitizerComponent ci,
   RgnHandle clipRegion
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

clipRegion

A handle to a `MacRegion` structure that defines the clipping region.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

VDSetClipState

Controls whether clipping is enabled.

```
VideoDigitizerError VDSetClipState (
   VideoDigitizerComponent ci,
   short clipEnable
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

clipEnable

Controls whether clipping is enabled. Place 0 into the short integer if clipping is disabled, and 1 if it is enabled.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

VDSetCompression

Specifies certain compression parameters.

```
VideoDigitizerError VDSetCompression (
   VideoDigitizerComponent ci,
   OSType compressType,
   short depth,
   Rect *bounds,
   CodecQ spatialQuality,
   CodecQ temporalQuality,
   long keyFrameRate
);
```

Parameters

ci

Identifies the application's connection to the video digitizer component. An application obtains this value from `OpenComponent` or `OpenDefaultComponent`.

compressType

A compressor type. This value corresponds to the `component` subtype of the compressor component; see `Codec Identifiers`.

depth

The depth at which the image is likely to be viewed. Compressors may use this as an indication of the color or grayscale resolution of the image. Values of 1, 2, 4, 8, 16, 24, and 32 indicate the number of bits per pixel for color images. Values of 33, 34, 36, and 40 correspond to 1-bit, 2-bit, 4-bit, and 8-bit grayscale images.

bounds

A pointer to a `Rect` structure that defines the desired boundaries of the compressed image.

spatialQuality

A constant (see below) that indicates the desired image quality for each frame in the sequence. See these constants:

`codecMinQuality`

`codecLowQuality`

`codecNormalQuality`

`codecHighQuality`

`codecMaxQuality`

`codecLosslessQuality`

temporalQuality

A constant (see below) that indicates the desired temporal quality for the sequence as a whole.

keyFrameRate

The maximum number of frames to allow between key frames. This value defines the minimum rate at which key frames are to appear in the compressed sequence; however, the video digitizer may insert key frames more often than an application specifies. If the application requests no temporal compression (that is, the application set the `temporalQuality` parameter to 0), the video digitizer ignores this parameter.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

VDSetCompressionOnOff

Allows an application to start and stop compression by video digitizers that can deliver either compressed or uncompressed image data.

```
VideoDigitizerError VDSetCompressionOnOff (
   VideoDigitizerComponent ci,
   Boolean state
);
```

Parameters

ci

Identifies the application's connection to the video digitizer component. An application obtains this value from `OpenComponent` or `OpenDefaultComponent`.

state

A Boolean value that indicates whether to enable or disable compression. Applications set this parameter to TRUE to enable compression. Setting it to FALSE disables compression.

Return Value

Digitizers that only provide compressed data have their `digiOutDoesCompressOnly` flag set to 1, rather than 0. These digitizers may either ignore this function or return a nonzero result code. See `Error Codes`. Return `noErr` if there is no error.

Discussion

Applications must call this function before they call either `VDSetCompression` (page 2320) or `VDCompressOneFrameAsync` (page 2274). This allows the video digitizer to prepare for the operation.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

VDSetContrast

Sets the current contrast value.

```
VideoDigitizerError VDSetContrast (
   VideoDigitizerComponent ci,
   unsigned short *contrast
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

contrast

A pointer to an integer that contains the new contrast value. The contrast value ranges from 0 to 65,535, where 0 represents no change to the basic image and larger values increase the contrast of the video image (they increase the slope of the transform). The digitizer component returns the new value, so that the application can avoid using unsupported values in future requests.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

VDSetDataRate

Instructs your video digitizer component to limit the rate at which it delivers compressed, digitized video data.

```
VideoDigitizerError VDSetDataRate (
   VideoDigitizerComponent ci,
   long bytesPerSecond
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

bytesPerSecond

The maximum data rate requested by the application, in bytes per second. This parameter is set to 0 to remove any data-rate restrictions.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This function is valid only for video digitizer components that can deliver compressed video; that is, components that support the `VDCompressOneFrameAsync` (page 2274) function. Components that support data-rate limiting set the `codecInfoDoesRateConstrain` flag to 1 in the `compressFlags` field of the `VDCompressionList` structure returned by the component in response to the `VDGetCompressionTypes` (page 2280) function. Your video digitizer component should return this data-rate limit in the `bytesPerSecond` parameter of the existing `VDGetDataRate` (page 2282) function.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

VDSetDestinationPort

Sets the destination port for a video digitizer.

```
VideoDigitizerError VDSetDestinationPort (
   VideoDigitizerComponent ci,
   CGrafPtr destPort
);
```

Parameters

ci

Specifies the video digitizer component for this operation. Applications can obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

destPort

A pointer to a `CGrafPort` structure.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

VDSetDigitizerRect

Sets the current video digitizer rectangle.

```
VideoDigitizerError VDSetDigitizerRect (
   VideoDigitizerComponent ci,
   Rect *digitizerRect
);
```

Parameters

`ci`
The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

`digitizerRect`
A pointer to a `Rect` structure that contains the size and location information for the digitizer rectangle. The coordinates of this rectangle must be relative to the maximum source rectangle. In addition, the digitizer rectangle must be within the maximum source rectangle. For a discussion of the relationship between these rectangles, see "Video Digitizer Components" in *Inside Macintosh: QuickTime* Components.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
All video digitizer components must support this function.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

VDSetDigitizerUserInterrupt

Sets custom interrupt functions.

```
VideoDigitizerError VDSetDigitizerUserInterrupt (
   VideoDigitizerComponent ci,
   long flags,
   VdigIntUPP userInterruptProc,
   long refcon
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

flags

Indicates when the interrupt function is to be called. If bit 0 is set to 1, the video digitizer component calls the custom interrupt procedure each time it starts to display an even-line field. If bit 1 is set to 1, the video digitizer component calls the custom interrupt procedure each time it starts to display an odd-line field. Applications may set both bits to 1.

userInterruptProc

A Universal Procedure Pointer to a `VdigIntProc` callback. Applications can set this parameter to `NIL` to remove a `VdigIntProc` callback.

refcon

Contains parameter data that is appropriate for the `callback`. Use this parameter to point to a data structure containing any information your callback needs.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

VDSetFieldPreference

Specifies which field to use in cases where the vertical scaling is less than half size.

```
VideoDigitizerError VDSetFieldPreference (
   VideoDigitizerComponent ci,
   short fieldFlag
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

fieldFlag

A constant (see below) that indicates which field to use. See these constants:

`vdUseAnyField`

`vdUseOddField`

`vdUseEvenField`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

All video digitizer components must support this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

VDSetFrameRate

Indicates an application's desired frame rate to the video digitizer.

```
VideoDigitizerError VDSetFrameRate (
   VideoDigitizerComponent ci,
   Fixed framesPerSecond
);
```

Parameters

ci

Identifies the application's connection to the video digitizer component. An application obtains this value from `OpenComponent` or `OpenDefaultComponent`.

framesPerSecond

The application's desired frame rate. Applications may set this parameter to 0 to return the digitizer to its default frame rate (typically 29.97 frames per second).

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

VDSetHue

Sets the current hue value.

```
VideoDigitizerError VDSetHue (
   VideoDigitizerComponent ci,
   unsigned short *hue
);
```

Parameters

ci
: The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

hue
: A pointer to an integer that contains the new hue value. Hue is similar to the tint control on a television, and it is specified in degrees with complementary colors set 180 degrees apart (red is 0 degrees, green is +120 degrees, and blue is -120 degrees). Video digitizer components support hue values that range from 0 (-180 degrees shift in hue) to 65,535 (+179 degrees shift in hue), where 32,767 represents a 0 degree shift in hue. The digitizer component returns the new value, so that the application can avoid using unsupported values in future requests.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

VDSetInput

Selects the input video source for a video digitizer component.

```
VideoDigitizerError VDSetInput (
   VideoDigitizerComponent ci,
   short input
);
```

Parameters

ci
: The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

input
: The input video source for this request. Video digitizer components number video sources sequentially, starting at 0. To request the first video source, an application sets this parameter to 0.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
All video digitizer components must support this function.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

VDSetInputColorSpaceMode

Chooses between color and grayscale digitized video.

```
VideoDigitizerError VDSetInputColorSpaceMode (
   VideoDigitizerComponent ci,
   short colorSpaceMode
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

colorSpaceMode

Controls color digitization. Set to 0 for grayscale, 1 for color.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

VDSetInputGammaRecord

Changes the active input gamma data structure.

```
VideoDigitizerError VDSetInputGammaRecord (
   VideoDigitizerComponent ci,
   VDGamRecPtr inputGammaPtr
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

inputGammaPtr

A `VDGammaRecord` structure.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

VDSetInputGammaValue

Sets the gamma values.

```
VideoDigitizerError VDSetInputGammaValue (
   VideoDigitizerComponent ci,
   Fixed channel1,
   Fixed channel2,
   Fixed channel3
);
```

Parameters

`ci`
: The video digitizer component for the request. Applications obtain this reference from the Component Manager's `OpenComponent` function.

`channel1`
: The gamma value for the red component of the input video signal.

`channel2`
: The gamma value for the green component of the input video signal.

`channel3`
: The gamma value for the blue component of the input video signal.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
These gamma values control the brightness of the input video signal. Your application can implement special color effects, such as turning off specific color channels, by calling this function.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

VDSetInputStandard

Specifies the input signaling standard to digitize.

```
VideoDigitizerError VDSetInputStandard (
   VideoDigitizerComponent ci,
   short inputStandard
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

inputStandard

A short integer (see below) that identifies the input signaling standard. See these constants:

`ntscIn`

`palIn`

`secamIn`

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

All video digitizer components must support this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

VDSetKeyColor

Sets the key color for video digitizing.

```
VideoDigitizerError VDSetKeyColor (
   VideoDigitizerComponent ci,
   long index
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

index

The new key color. This value must correspond to a color in the current color lookup table.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

All video digitizer components that support key colors must support this function.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

VDSetKeyColorRange

Defines a key color range for video digitizing.

```
VideoDigitizerError VDSetKeyColorRange (
   VideoDigitizerComponent ci,
   RGBColor *minRGB,
   RGBColor *maxRGB
);
```

Parameters

ci
: The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

minRGB
: A pointer to a field that contains the lower bound of the key color range. All colors in the color table between the color specified by the `minRGB` parameter and the color specified by the `maxRGB` parameter are considered key colors.

maxRGB
: A pointer to a field that contains the upper bound of the key color range. All colors in the color table between the color specified by the `minRGB` parameter and the color specified by the `maxRGB` parameter are considered key colors.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

VDSetMasterBlendLevel

Sets the blend level value for the input video signal.

```
VideoDigitizerError VDSetMasterBlendLevel (
   VideoDigitizerComponent ci,
   unsigned short *blendLevel
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

blendLevel

A pointer to a field that specifies the new master blend level. Valid values range from 0 to 65,535, where 0 corresponds to no video and 65,535 corresponds to all video. The digitizer component returns the new value in this field, so your application can avoid using unsupported values in future requests.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

VDSetPlayThruDestination

Establishes the destination settings for a video digitizer component.

```
VideoDigitizerError VDSetPlayThruDestination (
   VideoDigitizerComponent ci,
   PixMapHandle dest,
   RectPtr destRect,
   MatrixRecordPtr m,
   RgnHandle mask
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

dest

A handle to the destination `PixMap` structure. This pixel map may be in the video frame buffer of the Macintosh computer, or it may specify an offscreen buffer.

destRect

A pointer to a `Rect` structure that specifies the size and location of the video destination. This rectangle must be in the coordinate system of the destination `PixMap` structure specified by the `dest` parameter.

m

A pointer to a `MatrixRecord` structure containing the transformation matrix for the destination video image. To determine the capabilities of a video digitizer component, you can call `VDGetDigitizerInfo` (page 2284) in your application.

mask

A handle to a `MacRegion` structure that defines a mask. Applications can use masks to control clipping of the video into the destination rectangle. This mask region is defined in the destination coordinate space.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

All video digitizer components must support this function.

Special Considerations

Applications set the source digitizer rectangle by calling `VDSetDigitizerRect` (page 2324).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

VDSetPlayThruGlobalRect

Establishes the destination settings for a video digitizer component that is to digitize into a global rectangle.

```
VideoDigitizerError VDSetPlayThruGlobalRect (
   VideoDigitizerComponent ci,
   GrafPtr theWindow,
   Rect *globalRect
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

theWindow

A pointer to the destination window.

globalRect

A pointer to a `Rect` structure that specifies the size and location of the video destination. This rectangle must be in the coordinate system of the destination window specified by the `theWindow` parameter.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

The application provides the desired settings as parameters to this function. Not all video digitizer components support global rectangles.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

VDSetPlayThruOnOff

Controls continuous digitization.

```
VideoDigitizerError VDSetPlayThruOnOff (
   VideoDigitizerComponent ci,
   short state
);
```

Parameters

`ci`
The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

`state`
A short integer (see below) that specifies whether to use continuous digitization. When an application stops continuous digitization, the video digitizer component must restore its alpha channel, blending mask, or key color settings to graphics mode. See these constants:
`vdPlayThruOff`
`vdPlayThruOn`

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
When opened, video digitizer components are always set to off, so that no digitization is taking place. Your application can use this function to turn continuous digitization on and off.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

VDSetPLLFilterType

Specifies which phase locked loop (PLL) is to be active.

```
VideoDigitizerError VDSetPLLFilterType (
   VideoDigitizerComponent ci,
   short pllType
);
```

Parameters

`ci`
The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

pllType

Indicates which PLL is to be active. Values are 0 for broadcast mode and 1 for videotape recorder mode.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

VDSetPreferredImageDimensions

Sets the preferred image dimensions for a video digitizer.

```
VideoDigitizerError VDSetPreferredImageDimensions (
   VideoDigitizerComponent ci,
   long width,
   long height
);
```

Parameters

ci

Specifies the video digitizer component for this operation. Applications can obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

width

The preferred image width.

height

The preferred image height.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

VDSetPreferredPacketSize

Sets the preferred packet size for video digitizing.

```
VideoDigitizerError VDSetPreferredPacketSize (
   VideoDigitizerComponent ci,
   long preferredPacketSizeInBytes
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

preferredPacketSizeInBytes

The preferred packet size in bytes.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

This function was added in QuickTime 2.5 to support videoconferencing applications.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

VDSetSaturation

Sets the saturation value.

```
VideoDigitizerError VDSetSaturation (
   VideoDigitizerComponent ci,
   unsigned short *saturation
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

saturation

A pointer to an integer that contains the new saturation value. The saturation value controls color intensity. For example, at high saturation levels, red appears to be red; at low saturation, red appears as pink. Valid saturation values range from 0 to 65,535, where 0 is the minimum saturation value and 65,535 specifies maximum saturation. The video digitizer component attempts to set the saturation value to the value specified by this parameter. The digitizer component returns the new value, so that the application can avoid using unsupported values in future requests.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

VDSetSharpness

Sets the sharpness value.

```
VideoDigitizerError VDSetSharpness (
   VideoDigitizerComponent ci,
   unsigned short *sharpness
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

sharpness

A pointer to an integer that contains the new sharpness value. The sharpness value ranges from 0 to 65,535, where 0 represents no sharpness filtering and 65,535 represents full sharpness filtering. Higher values result in a visual impression of increased picture sharpness. The video digitizer component attempts to set the sharpness value to the value specified by this parameter. The digitizer component returns the new value, so that the application can avoid using unsupported values in future requests.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

VDSetTimeBase

Establishes the video digitizer's time coordinate system.

```
VideoDigitizerError VDSetTimeBase (
   VideoDigitizerComponent ci,
   TimeBase t
);
```

Parameters

ci

Identifies the application's connection to the video digitizer component. An application obtains this value from `OpenComponent` or `OpenDefaultComponent`.

t

A time base identifier. You can get this value from `NewTimeBase` (page 261).

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

VDSetupBuffers

Defines output buffers for use with asynchronous grabs.

```
VideoDigitizerError VDSetupBuffers (
   VideoDigitizerComponent ci,
   VdigBufferRecListHandle bufferList
);
```

Parameters

ci
The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

bufferList
A handle to a `VdigBufferRecList` structure. Video digitizer components extract information about the spatial characteristics of the video destinations from these buffers.

Return Value
See `Error Codes`. Returns `noErr` if there is no error.

Discussion
If you are developing a video digitizer component, note that the `matrix` field in the buffer list structure contains a pointer to the `MatrixRecord` structure. It is your responsibility to copy that matrix structure.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

VDSetWhiteLevelValue

Sets the white level value.

```
VideoDigitizerError VDSetWhiteLevelValue (
   VideoDigitizerComponent ci,
   unsigned short *whiteLevel
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

whiteLevel

A pointer to an integer that contains the new white level value. White level values range from 0 to 65,535, where 0 represents the minimum white value and 65,535 represents the maximum white value.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

VDUseSafeBuffers

Instructs a video digitizer to use protected buffers.

```
VideoDigitizerError VDUseSafeBuffers (
   VideoDigitizerComponent ci,
   Boolean useSafeBuffers
);
```

Parameters

ci

Specifies the video digitizer component for this operation. Applications can obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

useSafeBuffers

Pass TRUE to use protected buffers; pass FALSE otherwise.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

VDUseThisCLUT

Specifies the lookup table for color digitization.

```
VideoDigitizerError VDUseThisCLUT (
   VideoDigitizerComponent ci,
   CTabHandle colorTableHandle
);
```

Parameters

ci

The video digitizer component for the request. Applications obtain this reference from `OpenComponent` or `OpenDefaultComponent`.

colorTableHandle

A handle to a `ColorTable` structure. The video digitizer component uses the color table referred to by this parameter.

Return Value

See `Error Codes`. Returns `noErr` if there is no error.

Discussion

This feature is useful only for capturing 8-bit color video.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`QuickTimeComponents.h`

Callbacks

Data Types

DigitizerInfo

Contains information about the capabilities and current status of a video digitizer component.

```
struct DigitizerInfo {
     short       vdigType;
     long        inputCapabilityFlags;
     long        outputCapabilityFlags;
     long        inputCurrentFlags;
     long        outputCurrentFlags;
     short       slot;
     GDHandle    gdh;
     GDHandle    maskgdh;
     short       minDestHeight;
     short       minDestWidth;
     short       maxDestHeight;
     short       maxDestWidth;
     short       blendLevels;
     long        reserved;
 };
```

Fields

`vdigType`

Discussion

Constant (see below) that specifies the type of video digitizer component. See these constants:

`vdTypeBasic`

`vdTypeAlpha`

`vdTypeMask`

`vdTypeKey`

`inputCapabilityFlags`

Discussion

Constant (see below) that specifies the capabilities of the video digitizer component with respect to the input video signal. See these constants:

`digiInDoesNTSC`

`digiInDoesPAL`

`digiInDoesSECAM`

`digiInDoesGenLock`

`digiInDoesComposite`

`digiInDoesComponent`

`digiInVTR_Broadcast`

`digiInDoesColor`

`digiInDoesBW`

`outputCapabilityFlags`

Discussion

Constant (see below) that specifies the capabilities of the video digitizer component with respect to the output digitized video information. See these constants:

```
    digiOutDoes1
    digiOutDoes2
    digiOutDoes4
    digiOutDoes8
    digiOutDoes16
    digiOutDoes32
    digiOutDoesDither
    digiOutDoesStretch
    digiOutDoesShrink
    digiOutDoesMask
    digiOutDoesDouble
    digiOutDoesQuad
    digiOutDoesQuarter
    digiOutDoesSixteenth
    digiOutDoesRotate
    digiOutDoesHorizFlip
    digiOutDoesVertFlip
    digiOutDoesSkew
    digiOutDoesBlend
    digiOutDoesWarp
    digiOutDoesHWPlayThru
    digiOutDoesILUT
    digiOutDoesKeyColor
    digiOutDoesAsyncGrabs
    digiOutDoesUnreadableScreenBits
    digiOutDoesCompress
    digiOutDoesCompressOnly
    digiOutDoesPlayThruDuringCompress
```

`inputCurrentFlags`

Discussion

Specifies the current status of the video digitizer with respect to the input video signal. Video digitizer components report their current input status by returning a flags field that contains 1 bit for each of the applicable `inputCapabilityFlags` constants (see below), plus additional `inputCurrentFlags` constants (see below) as appropriate. The digitizer component sets these flags to reflect its current status. When reporting input status, for example, a video digitizer component sets the `digiInDoesGenLock` flag to 1 whenever the digitizer component is deriving its time signal from the input video. When reporting its input capabilities, the digitizer component sets this flag to 1 to indicate that it can derive its timing from the input video. See these constants:

```
    digiInSignalLock
```

`outputCurrentFlags`

Discussion
Specifies the current status of the video digitizer with respect to the output video signal. Video digitizer components report their current output status by returning a flags field that contains 1 bit for each of the applicable `outputCapabilityFlags` constants (see below)

`slot`

Discussion
Identifies the slot that contains the video digitizer interface card.

`gdh`

Discussion
Contains a handle to the graphics device that defines the screen to which the digitized data is to be written. Set this field to `NIL` if your application is not constrained to a particular graphics device.

`maskgdh`

Discussion
Contains a handle to the graphics device that contains the `mask` plane. This field is used only by digitizers that clip by means of mask planes.

`minDestHeight`

Discussion
Indicates the smallest height value the digitizer component can accommodate in its destination.

`minDestWidth`

Discussion
Indicates the smallest width value the digitizer component can accommodate in its destination.

`maxDestHeight`

Discussion
Indicates the largest height value the digitizer component can accommodate in its destination.

`maxDestWidth`

Discussion
Indicates the largest width value the digitizer component can accommodate in its destination.

`blendLevels`

Discussion
Specifies the number of blend levels the video digitizer component supports.

`reserved`

Discussion
Reserved. Set this field to 0.

Discussion
Your application can retrieve information about the capabilities and current status of a video digitizer component. You call `VDGetDigitizerInfo` (page 2284) to retrieve all this information from a video digitizer component. In response, the component formats a `DigitizerInfo` structure. The contents of this structure fully define the capabilities and current status of the video digitizer component.

Related Functions
`VDGetDigitizerInfo` (page 2284)

Declared In
`QuickTimeComponents.h`

GrafPort

Defines a complete drawing environment for black-and-white graphics operations.

```
struct GrafPort {
     short          device;
     BitMap         portBits;
     Rect           portRect;
     RgnHandle      visRgn;
     RgnHandle      clipRgn;
     Pattern        bkPat;
     Pattern        fillPat;
     Point          pnLoc;
     Point          pnSize;
     short          pnMode;
     Pattern        pnPat;
     short          pnVis;
     short          txFont;
     StyleField     txFace;
     short          txMode;
     short          txSize;
     Fixed          spExtra;
     long           fgColor;
     long           bkColor;
     short          colrBit;
     short          patStretch;
     Handle         picSave;
     Handle         rgnSave;
     Handle         polySave;
     QDProcsPtr     grafProcs;
 };
```

Fields

`device`

Discussion
See `CGrafPort`.

`portBits`

Discussion
See `CGrafPort`. In a `GrafPort` structure, this field contains a complete 14-byte `BitMap` structure. In a `CGrafPort` structure, this field is partly replaced by the 4-byte `portPixMap` field, which contains a handle to a `PixMap` structure. In what would be the `rowBytes` field of the `BitMap` structure, a `CGrafPort` structure has a 2-byte `portVersion` field in which the two high bits are always set to 1. QuickTime uses these bits to distinguish `CGrafPort` records from `GrafPort` records, in which the two high bits of the `rowBytes` field are always 0. Following the `portBits` field in the `CGrafPort` structure are the `portVersion` and `grafVars` fields. The `grafVars` field contains a handle to a `GrafVars` structure; this handle is not included in the `GrafPort` structure. For information about the `GrafVars` structure, see *Inside Macintosh: Imaging With QuickDraw*.

`portRect`

Discussion
See `CGrafPort`.

`visRgn`

Discussion
See `CGrafPort`.

`clipRgn`

Discussion
See `CGrafPort`.

`bkPat`

Discussion
In a `GrafPort` structure, the `bkPat`, `pnPat`, and `fillPat` fields hold 8-byte bit patterns. In a `CGrafPort` structure, these fields are partly replaced by three 4-byte handles to pixel patterns. The resulting 12 bytes of additional space in the `CGrafPort` structure are taken up by the `rgbFgColor` and `rgbBkColor` fields, which contain 6-byte `RGBColor` structures specifying the optimal foreground and background colors for the color graphics port. Note that the closest matching available colors, which QuickTime actually uses to render the foreground and background, are stored in the `fgColor` and `bkColor` fields of the `CGrafPort` structure.

`fillPat`

Discussion
See the `bkPat` field (above).

`pnLoc`

Discussion
See `CGrafPort`.

`pnSize`

Discussion
See `CGrafPort`.

`pnMode`

Discussion
See `CGrafPort`.

`pnPat`

Discussion
See the `bkPat` field (above).

`pnVis`

Discussion
See `CGrafPort`.

`txFont`

Discussion
See `CGrafPort`.

`txFace`

Discussion
The character `style` of the text, with values from the set defined by the `Style` type, which includes such styles as bold, italic, and shaded. You can apply stylistic variations either alone or in combination. This field is initially set to plain text.

`txMode`

Discussion
See `CGrafPort`.

`txSize`

Discussion
See `CGrafPort`.

`spExtra`

Discussion
See `CGrafPort`.

`fgColor`

Discussion
See `CGrafPort`.

`bkColor`

Discussion
See `CGrafPort`.

`colrBit`

Discussion
See `CGrafPort`.

`patStretch`

Discussion
See `CGrafPort`.

`picSave`

Discussion
See `CGrafPort`.

`rgnSave`

Discussion
See `CGrafPort`.

`polySave`

Discussion
See `CGrafPort`.

`grafProcs`

Discussion
See `CGrafPort`. In a `GrafPort` structure, you can supply this field with a pointer to a `QDProcs` structure; in a `CGrafPort` structure, you provide this field with a pointer to a `CQDProcs` structure.

Discussion
See `CGrafPort`.

Version Notes
The `GrafPort` structure has been largely superseded by the `CGrafPort` structure, which defines a full color environment. The two structures are the same size; QuickTime distinguishes between them by examining the `portBits` field.

Declared In
`QuickTimeComponents.h`

GrafPtr

Represents a type used by the Video Components API.

```
typedef GrafPort * GrafPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

QTVideoOutputComponent

Represents a type used by the Video Components API.

```
typedef ComponentInstance QTVideoOutputComponent;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

RectPtr

Represents a type used by the Video Components API.

```
typedef Rect * RectPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`IOMacOSTypes.h`

VDCompressionListHandle

Represents a type used by the Video Components API.

```
typedef VDCompressionListPtr * VDCompressionListHandle;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

VDCompressionListPtr

Represents a type used by the Video Components API.

```
typedef VDCompressionList * VDCompressionListPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

VDGammaRecord

Holds a gamma table.

```
struct VDGammaRecord {
     Ptr    csGTable;
 };
```

Fields
`csGTable`

Discussion
A pointer to a gamma table.

Declared In
`QuickTimeComponents.h`

VDGamRecPtr

Represents a type used by the Video Components API.

```
typedef VDGammaRecord * VDGamRecPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`IOMacOSVideo.h`

VdigBufferRecListHandle

Represents a type used by the Video Components API.

```
typedef VdigBufferRecListPtr * VdigBufferRecListHandle;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

VdigBufferRecListPtr

Represents a type used by the Video Components API.

```
typedef VdigBufferRecList * VdigBufferRecListPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

VideoDigitizerComponent

Represents a type used by the Video Components API.

```
typedef ComponentInstance VideoDigitizerComponent;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

VideoDigitizerError

Represents a type used by the Video Components API.

```
typedef ComponentResult VideoDigitizerError;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickTimeComponents.h`

Constants

compositeIn

Constants grouped with compositeIn.

```
enum {
  compositeIn                   = 0,    /* input is composite format */
  sVideoIn                      = 1,    /* input is sVideo format */
  rgbComponentIn                = 2,    /* input is rgb component format */
  rgbComponentSyncIn            = 3,    /* input is rgb component format (sync on
 green?)*/
  yuvComponentIn                = 4,    /* input is yuv component format */
  yuvComponentSyncIn            = 5,    /* input is yuv component format (sync on
 green?) */
  tvTunerIn                     = 6,
  sdiIn                         = 7
};
```

Declared In
`QuickTimeComponents.h`

Video Digitizer Capabilities

Flags that indicate the input and output capabilities of a video digitizer.

```
enum {
  digiInDoesNTSC                = 1L << 0, /* digitizer supports NTSC input format
 */
  digiInDoesPAL                 = 1L << 1, /* digitizer supports PAL input format
 */
  digiInDoesSECAM               = 1L << 2, /* digitizer supports SECAM input format
 */
  digiInDoesGenLock             = 1L << 7, /* digitizer does genlock */
  digiInDoesComposite           = 1L << 8, /* digitizer supports composite input
type */
  digiInDoesSVideo              = 1L << 9, /* digitizer supports S-Video input type
 */
  digiInDoesComponent           = 1L << 10, /* digitizer supports component = rgb,
 input type */
  digiInVTR_Broadcast           = 1L << 11, /* digitizer can differentiate between
 the two */
  digiInDoesColor               = 1L << 12, /* digitizer supports color */
  digiInDoesBW                  = 1L << 13, /* digitizer supports black & white */
                                        /* Digitizer Input Current Flags = these
are valid only during active operating conditions,   */
  digiInSignalLock              = 1L << 31 /* digitizer detects input signal is
locked, this bit = horiz lock || vertical lock */
};
enum {
  digiOutDoes1                  = 1L << 0, /* digitizer supports 1 bit pixels */
  digiOutDoes2                  = 1L << 1, /* digitizer supports 2 bit pixels */
  digiOutDoes4                  = 1L << 2, /* digitizer supports 4 bit pixels */
  digiOutDoes8                  = 1L << 3, /* digitizer supports 8 bit pixels */
  digiOutDoes16                 = 1L << 4, /* digitizer supports 16 bit pixels */
  digiOutDoes32                 = 1L << 5, /* digitizer supports 32 bit pixels */
  digiOutDoesDither             = 1L << 6, /* digitizer dithers in indexed modes
*/
  digiOutDoesStretch            = 1L << 7, /* digitizer can arbitrarily stretch */
  digiOutDoesShrink             = 1L << 8, /* digitizer can arbitrarily shrink */
  digiOutDoesMask               = 1L << 9, /* digitizer can mask to clipping regions
 */
  digiOutDoesDouble             = 1L << 11, /* digitizer can stretch to exactly
double size */
  digiOutDoesQuad               = 1L << 12, /* digitizer can stretch exactly
quadruple size */
  digiOutDoesQuarter            = 1L << 13, /* digitizer can shrink to exactly
quarter size */
  digiOutDoesSixteenth          = 1L << 14, /* digitizer can shrink to exactly
sixteenth size */
  digiOutDoesRotate             = 1L << 15, /* digitizer supports rotate
transformations */
  digiOutDoesHorizFlip          = 1L << 16, /* digitizer supports horizontal flips
 Sx < 0 */
  digiOutDoesVertFlip           = 1L << 17, /* digitizer supports vertical flips
Sy < 0 */
  digiOutDoesSkew               = 1L << 18, /* digitizer supports skew = shear,twist,
 */
  digiOutDoesBlend              = 1L << 19,
  digiOutDoesWarp               = 1L << 20,
  digiOutDoesHW_DMA             = 1L << 21, /* digitizer not constrained to local
 device */
  digiOutDoesHWPlayThru         = 1L << 22, /* digitizer doesn't need time to play
 thru */
```

```
  digiOutDoesILUT              = 1L << 23, /* digitizer does inverse LUT for index
 modes */
  digiOutDoesKeyColor           = 1L << 24, /* digitizer does key color functions
 too */
  digiOutDoesAsyncGrabs         = 1L << 25, /* digitizer supports async grabs */
  digiOutDoesUnreadableScreenBits = 1L << 26, /* playthru doesn't generate readable
 bits on screen*/
  digiOutDoesCompress          = 1L << 27, /* supports alternate output data types
 */
  digiOutDoesCompressOnly       = 1L << 28, /* can't provide raw frames anywhere
*/
  digiOutDoesPlayThruDuringCompress = 1L << 29, /* digi can do playthru while
providing compressed data */
  digiOutDoesCompressPartiallyVisible = 1L << 30, /* digi doesn't need all bits
visible on screen to do hardware compress */
  digiOutDoesNotNeedCopyOfCompressData = 1L << 31 /* digi doesn't need any
bufferization when providing compressed data */
};
```

Constants

`digiInDoesNTSC`

The video digitizer supports National Television System Committee (NTSC) format input video signals. This flag is set to 1 if the digitizer component supports NTSC video.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeComponents.h`.

`digiInDoesPAL`

The video digitizer component supports Phase Alternation Line (PAL) format input video signals. This flag is set to 1 if the digitizer component supports PAL video.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeComponents.h`.

`digiInDoesSECAM`

The video digitizer component supports Systeme Electronique Couleur avec Memoire (SECAM) format input video signals. This flag is set to 1 if the digitizer component supports SECAM video.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeComponents.h`.

`digiInDoesGenLock`

The video digitizer component supports genlock; that is, the digitizer can derive its timing from an external time base. This flag is set to 1 if the digitizer component supports genlock.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeComponents.h`.

`digiInDoesComposite`

The video digitizer component supports composite input video. This flag is set to 1 if the digitizer component supports composite input.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeComponents.h`.

`digiInDoesComponent`
: The video digitizer component supports RGB input video. This flag is set to 1 if the digitizer component supports RGB input.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`digiInVTR_Broadcast`
: The video digitizer component can distinguish between an input signal that emanates from a videotape player and a broadcast signal. This flag is set to 1 if the digitizer component can differentiate between the two different signal types.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`digiInDoesColor`
: The video digitizer component supports color input. This flag is set to 1 if the digitizer component can accept color input.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`digiInDoesBW`
: The video digitizer component supports grayscale input. This flag is set to 1 if the digitizer component can accept grayscale input.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`digiInSignalLock`
: The video digitizer component is locked onto the input signal. If this flag is set to 1, the digitizer component detects either vertical or horizontal signal lock.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`digiOutDoes1`
: The video digitizer component can work with pixel maps that contain 1-bit pixels. If this flag is set to 1, then the digitizer component can write images that contain 1-bit pixels. If this flag is set to 0, then the digitizer component cannot handle such images.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`digiOutDoes2`
: The video digitizer component can work with pixel maps that contain 2-bit pixels. If this flag is set to 1, then the digitizer component can write images that contain 2-bit pixels. If this flag is set to 0, then the digitizer component cannot handle such images.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`digiOutDoes4`
: The video digitizer component can work with pixel maps that contain 4-bit pixels. If this flag is set to 1, then the digitizer component can write images that contain 4-bit pixels. If this flag is set to 0, then the digitizer component cannot handle such images.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`digiOutDoes8`
: The video digitizer component can work with pixel maps that contain 8-bit pixels. If this flag is set to 1, then the digitizer component can write images that contain 8-bit pixels. If this flag is set to 0, then the digitizer component cannot handle such images.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`digiOutDoes16`
: The video digitizer component can work with pixel maps that contain 16-bit pixels. If this flag is set to 1, then the digitizer component can write images that contain 16-bit pixels. If this flag is set to 0, then the digitizer component cannot handle such images.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`digiOutDoes32`
: The video digitizer component can work with pixel maps that contain 32-bit pixels. If this flag is set to 1, then the digitizer component can write images that contain 32-bit pixels. If this flag is set to 0, then the digitizer component cannot handle such images.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`digiOutDoesDither`
: The video digitizer component supports dithering. If this flag is set to 1, the component supports dithering of colors. If this flag is set to 0, the digitizer component does not support dithering.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`digiOutDoesStretch`
: The video digitizer component can stretch images to arbitrary sizes. If this flag is set to 1, the digitizer component can stretch images. If this flag is set to 0, the digitizer component does not support stretching.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`digiOutDoesShrink`
: The video digitizer component can shrink images to arbitrary sizes. If this flag is set to 1, the digitizer component can shrink images. If this flag is set to 0, the digitizer component does not support shrinking.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`digiOutDoesMask`
: The video digitizer component can handle clipping regions. If this flag is set to 1, the digitizer component can mask to an arbitrary clipping region. If this flag is set to 0, the digitizer component does not support clipping regions.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`digiOutDoesDouble`
: The video digitizer component supports stretching to quadruple size when displaying the output video. The parameters for the stretch operation are specified in the matrix structure for the request; the component modifies the scaling attributes of the matrix (see the chapter "Movie Toolbox" in Inside Macintosh: QuickTime for information about transformation matrices). If this flag is set to 1, the digitizer component can stretch an image to exactly four times its original size, up to the maximum size specified by the `maxDestHeight` and `maxDestWidth` fields in the digitizer information structure. If this flag is set to 0, the digitizer component does not support stretching to quadruple size.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`digiOutDoesQuad`
: The video digitizer component supports stretching an image to 16 times its original size when displaying the output video. The parameters for the stretch operation are specified in the matrix structure for the request; the component modifies the scaling attributes of the matrix (see the chapter "Movie Toolbox" in Inside Macintosh: QuickTime for information about transformation matrices). If this flag is set to 1, the digitizer component can stretch an image to exactly 16 times its original size, up to the maximum size specified by the `maxDestHeight` and `maxDestWidth` fields in the digitizer information structure. If this flag is set to 0, the digitizer component does not support this capability.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`digiOutDoesQuarter`
: The video digitizer component can shrink an image to one-quarter of its original size when displaying the output video. The parameters for the shrink operation are specified in the matrix structure for the request; the component modifies the scaling attributes of the matrix (see the chapter "Movie Toolbox" in Inside Macintosh: QuickTime for information about transformation matrices). If this flag is set to 1, the digitizer component can shrink an image to exactly one-quarter of its original size, down to the minimum size specified by the `minDestHeight` and `minDestWidth` fields in the digitizer information structure. If this flag is set to 0, the digitizer component does not support this capability.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`digiOutDoesSixteenth`
: The video digitizer component can shrink an image to 1/16 of its original size when displaying the output video. The parameters for the shrink operation are specified in the matrix structure for the request; the digitizer component modifies the scaling attributes of the matrix (see the chapter "Movie Toolbox" in Inside Macintosh: QuickTime for information about transformation matrices). If this flag is set to 1, the digitizer component can shrink an image to exactly 1/16 of its original size, down to the minimum size specified by the `minDestHeight` and `minDestWidth` fields in the digitizer information structure. If this flag is set to 0, the digitizer component does not support this capability.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`digiOutDoesRotate`
: The video digitizer component can rotate an image when displaying the output video. The parameters for the rotation are specified in the matrix structure for an operation. If this flag is set to 1, the digitizer component can rotate the image. If this flag is set to 0, the digitizer component cannot rotate the resulting image.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`digiOutDoesHorizFlip`
: The video digitizer component can flip an image horizontally when displaying the output video. The parameters for the horizontal flip are specified in the matrix structure for an operation. If this flag is set to 1, the digitizer component can flip the image. If this flag is set to 0, the digitizer component cannot flip the resulting image.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`digiOutDoesVertFlip`
: The video digitizer component can flip an image vertically when displaying the output video. The parameters for the vertical flip are specified in the matrix structure for an operation. If this flag is set to 1, the digitizer component can flip the image. If this flag is set to 0, the digitizer component cannot flip the resulting image.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`digiOutDoesSkew`
: The video digitizer component can skew an image when displaying the output video. Skewing an image distorts it linearly along only a single axis; for example, drawing a rectangular image into a parallelogram-shaped region. The parameters for the skew operation are specified in the matrix structure for the request. If this flag is set to 1, the digitizer component can skew an image. If this flag is set to 0, the digitizer component does not support this capability.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`digiOutDoesBlend`
: The video digitizer component can blend the resulting image with a matte when displaying the output video. The matte is provided by the application by defining either an alpha channel or a mask plane. If this flag is set to 1, the digitizer component can blend. If this flag is set to 0, the digitizer component does not support this capability.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`digiOutDoesWarp`
: The video digitizer component can warp an image when displaying the output video. Warping an image distorts it along one or more axes, perhaps nonlinearly, in effect "bending" the result region. The parameters for the warp operation are specified in the matrix structure for the request. If this flag is set to 1, the digitizer component can warp an image. If this flag is set to 0, the digitizer component does not support this capability.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`digiOutDoesHWPlayThru`
: The video digitizer component does not need idle time in order to display its video. If this flag is set to 1, your application does not need to grant processor time to the digitizer component at normal display speeds.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`digiOutDoesILUT`
: The video digitizer component supports inverse lookup tables for indexed color modes. If this flag is set to 1, the digitizer component uses inverse lookup tables when appropriate.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`digiOutDoesKeyColor`
: The video digitizer component supports clipping by means of key colors. If this flag is set to 1, the digitizer component can clip to a region defined by a key color.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`digiOutDoesAsyncGrabs`
: The video digitizer component can operate asynchronously. If this flag is set to 1, your application can use the `VDSetupBuffers` and `VDGrabOneFrameAsync` functions (described on page 0-669 and page 0-671, respectively).

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`digiOutDoesUnreadableScreenBits`
: The video digitizer may place pixels on the screen that cannot be used when compressing images.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`digiOutDoesCompress`
: The video digitizer component supports compressed source devices. These devices provide compressed data directly, without having to use the Image Compression Manager. See "Controlling Compressed Source Devices" beginning on page 0-657 for more information about the functions that applications can use to work with compressed source devices.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`digiOutDoesCompressOnly`
: The video digitizer component only provides compressed image data; the component cannot provide displayable data. This flag only applies to digitizers that support compressed source devices.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

`digiOutDoesPlayThruDuringCompress`
: The video digitizer component can draw images on the screen at the same time that it is delivering compressed image data. This flag only applies to digitizers that support compressed source devices.

  Available in Mac OS X v10.0 and later.

  Declared in `QuickTimeComponents.h`.

Declared In
`QuickTimeComponents.h`

currentIn

Constants grouped with currentIn.

```
enum {
  ntscIn                        = 0,    /* current input format */
  currentIn                     = 0,    /* ntsc input format */
  palIn                         = 1,    /* pal input format */
  secamIn                       = 2,    /* secam input format */
  ntscReallyIn                  = 3     /* ntsc input format */
};
```

Declared In
`QuickTimeComponents.h`

VDGetDeviceNameAndFlags Values

Constants passed to VDGetDeviceNameAndFlags.

```
enum {
  vdDeviceFlagShowInputsAsDevices = (1 << 0), /* Tell the Panel to promote Inputs
 to Devices*/
  vdDeviceFlagHideDevice        = (1 << 1) /* Omit this Device entirely from the
list*/
};
```

Declared In
`QuickTimeComponents.h`

vdFlagCaptureAlwaysUseTimeBase

Constants grouped with vdFlagCaptureAlwaysUseTimeBase.

```
enum {
  vdFlagCaptureStarting         = (1 << 0), /* Capture is about to start; allocate
 bandwidth */
  vdFlagCaptureStopping         = (1 << 1), /* Capture is about to stop; stop
queuing frames*/
  vdFlagCaptureIsForPreview    = (1 << 2), /* Capture is just to screen for preview
 purposes*/
  vdFlagCaptureIsForRecord      = (1 << 3), /* Capture is going to be recorded*/
  vdFlagCaptureLowLatency      = (1 << 4), /* Fresh frames are more important than
 delivering every frame - don't queue too much*/
  vdFlagCaptureAlwaysUseTimeBase = (1 << 5), /* Use the timebase for every frame;
 don't worry about making durations uniform*/
  vdFlagCaptureSetSettingsBegin = (1 << 6), /* A series of calls are about to be
made to restore settings.*/
  vdFlagCaptureSetSettingsEnd   = (1 << 7) /* Finished restoring settings; any set
 calls after this are from the app or UI*/
};
```

Declared In
`QuickTimeComponents.h`

VDSetPlayThruOnOff Values

Constants passed to VDSetPlayThruOnOff.

```
enum {
  vdPlayThruOff                  = 0,
  vdPlayThruOn                   = 1
};
```

Declared In
`QuickTimeComponents.h`

VdigType Values

Constants passed to VdigType.

```
enum {
  vdTypeBasic                    = 0,    /* basic, no clipping */
  vdTypeAlpha                    = 1,    /* supports clipping with alpha channel */
  vdTypeMask                     = 2,    /* supports clipping with mask plane */
  vdTypeKey                      = 3     /* supports clipping with key color(s) */
};
```

Constants

`vdTypeBasic`

Basic video digitizer; does not support any clipping.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeComponents.h`.

`vdTypeAlpha`

Supports clipping by means of an alpha channel.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeComponents.h`.

`vdTypeMask`

Supports clipping by means of a mask plane.

Available in Mac OS X v10.0 and later.

Declared in `QuickTimeComponents.h`.

Declared In
`QuickTimeComponents.h`

VDSetFieldPreference Values

Constants passed to VDSetFieldPreference.

```
enum {
  vdUseAnyField                  = 0,    /* Digitizers choice on field use */
  vdUseOddField                  = 1,    /* Use odd field for half size vert and
smaller */
  vdUseEvenField                 = 2     /* Use even field for half size vert and
smaller */
};
```

Declared In
`QuickTimeComponents.h`

Windows API Reference for QuickTime

Framework:	Frameworks/QuickTime.framework
Declared in	QTML.h

Overview

Apple provides a small set of utility functions that are used for developing QuickTime applications in the Windows environment.

Functions

QTMLCreateMutex

Creates a synchronization object to facilitate mutually exclusive access to a Windows data structure.

```
QTMLMutex QTMLCreateMutex (
   void
);
```

Discussion
This function creates a mutex object for guarded access to data structures and routines that require mutually exclusive access. In a multithreaded preemptive environment, such as Windows NT, you can use the various mutex utility functions such as `QTMLGrabMutex` (page 2362) to protect a shared resource from simultaneous access by multiple threads or processes. Mutex objects are used throughout QTML to provide such protection.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
audiocodec

audiocodec.win

QTCarbonCoreImage101

WhackedTV

Declared In
`QTML.h`

QTMLDestroyMutex

Deallocates a synchronization object created by QTMLCreateMutex.

```
void QTMLDestroyMutex (
   QTMLMutex mu
);
```

Parameters

mu

A mutex object.

Discussion

Call this function to deallocate the mutex object created by `QTMLCreateMutex` (page 2361).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

audiocodec

audiocodec.win

WhackedTV

Declared In

`QTML.h`

QTMLGrabMutex

Confers ownership of a mutex created by QTMLCreateMutex.

```
void QTMLGrabMutex (
   QTMLMutex mu
);
```

Parameters

mu

A mutex object.

Discussion

Call this function when you require exclusive ownership of the resource guarded by a mutex. This function will return when you have gained this ownership. In the case where another thread or process holds the mutex, this function waits until that process or thread relinquishes control. If you need to determine if you can grab the mutex, without actually grabbing it, call `QTMLTryGrabMutex` (page 2363).

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.3 and later.

Related Sample Code

audiocodec

audiocodec.win

QTCarbonCoreImage101

WhackedTV

Declared In
`QTML.h`

QTMLReturnMutex

Releases ownership of a QTMLMutex object.

```
void QTMLReturnMutex (
   QTMLMutex mu
);
```

Parameters

mu

A mutex object.

Discussion
Call this function to balance a call to `QTMLGrabMutex` (page 2362) when you are ready to relinquish control of the mutex and corresponding shared resource. By making this call, you allow other processes or threads waiting for the release of this mutex to gain access.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
audiocodec

audiocodec.win

QTCarbonCoreImage101

WhackedTV

Declared In
`QTML.h`

QTMLTryGrabMutex

Determines if you would be able to get immediate ownership of a mutex created by QTMLCreateMutex.

```
Boolean QTMLTryGrabMutex (
   QTMLMutex mu
);
```

Parameters

mu

A mutex object.

Return Value
Returns TRUE if you are able to immediately grab the mutex, via the `QTMLGrabMutex` (page 2362) call, without having to wait.

Discussion
Call this function when you need to preflight a `QTMLGrabMutex` (page 2362) call.

Special Considerations

Under normal circumstances you should not need to make this call.

Version Notes
Introduced in QuickTime 4.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`QTML.h`

QTMLYieldCPU

Yields time to other threads while your code is in a tight loop.

```
void QTMLYieldCPU (
   void
);
```

Discussion
Use this function from within tight loops to yield time to other threads. Using this function is similar to calling `SystemTask` from within a Macintosh event loop.

Version Notes
Introduced in QuickTime 3 or earlier.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`QTML.h`

QTMLYieldCPUTime

Yields time to other threads and specifies the sleep time while in a tight loop.

```
void QTMLYieldCPUTime (
   long milliSeconds,
   unsigned long flags
);
```

Parameters

milliSeconds
Number of milliseconds to sleep before returning to the caller.

flags
A flag (see below) that specifies an option for this function. See these constants:
`kQTMLHandlePortEvents`

Discussion

Use this function from within tight loops to yield time to other threads.

Special Considerations

This function differs from `QTMLYieldCPU` (page 2364) in that you can specify the time to sleep as well as optionally have QTML process Win32 messages while waiting for the yield time to expire.

Version Notes

Introduced in QuickTime 3 or earlier.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`QTML.h`

Callbacks

Data Types

QTMLMutex

Represents a type used by the Windows API API.

```
typedef long QTMLMutex;
```

Availability

Available in Mac OS X v10.3 and later.

Declared In

`QTML.h`

Constants

QTMLYieldCPUTime Values

Constants passed to QTMLYieldCPUTime.

```
enum {
  kQTMLHandlePortEvents          = (1L << 0) /* ask for event handling during the
yield*/
};
```

Declared In

`QTML.h`

QuickTime Atoms and Resources Reference

Framework:	Frameworks/QuickTime.framework
Declared in	QuickTime.h

Overview

This reference covers the API details of QuickTime atoms and public resource types.

QuickTime Atoms

Atoms

0x00000000 Terminates an audio atom list.

```
struct AudioTerminatorAtom {
    long        size;
    OSType      atomType;
};
```

Fields
`size`

Discussion
The size in bytes of this atom structure.

`atomType`

Discussion
Constant `kAudioTerminatorAtomType`.

Programming Info
C interface file: `Sound.h`

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

0x00000001 A sprite property matrix atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields
`atomType`

Discussion
0x00000001.

`data`

Discussion
The sprite matrix property, a structure of type `MatrixRecord`.

Parent Atom

`'sprt'` (page 2428)
Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

0x00000004 A sprite visible property atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields
`atomType`

Discussion
0x00000004.

`data`

Discussion
The sprite visible property, of type `short`.

Parent Atom
`'sprt'` (page 2428)
Parent atom can contain only one atom of this type.

See Also
For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

0x00000005 A sprite property layer atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields
`atomType`

Discussion
0x00000005.

`data`

Discussion
The sprite layer property, of type `short`.

Parent Atom
`'sprt'` (page 2428)
Parent atom can contain only one atom of this type.

See Also
For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

0x00000006 A sprite graphics mode property atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields
`atomType`

Discussion
0x00000006.

`data`

Discussion
The sprite graphics mode property.

Parent Atom

`'sprt'` (page 2428)

Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

0x00000064 A sprite image index atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields

`atomType`

Discussion

0x00000064.

`data`

Discussion

The sprite image index property, of type `short`.

Parent Atom

`'sprt'` (page 2428)

Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

0x00000065 A sprite background color property atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields

`atomType`

Discussion

0x00000065.

`data`

Discussion

The sprite background color property, a structure of type `RGBColor`.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

0x00000066 A sprite property offscreen bit depth atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields

`atomType`

Discussion

0x00000066.

`data`

Discussion
The sprite offscreen bit depth property, of type `short`.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

0x00000067 A sprite property sample format atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields
`atomType`

Discussion
0x00000067.

`data`

Discussion
The sprite sample format property, of type `short`.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'AllF' User data list entry atom to play all frames.

```
struct MoviesUserData {
    long    size;
    long    udType;
    char    data[1];
};
```

Fields
`size`

Discussion
The size in bytes of this atom structure.

`udType`

Discussion
`'AllF'`.

`data`

Discussion
A byte indicating that all frames of video should be played, regardless of timing.

Parent Atom

`'udta'` (page 2446)

Parent atom can contain only one atom of this type.

Programming Info
C interface file: `MoviesFormat.h`

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'beha' Defines sprite behavior.

This is a QT container atom; it is not declared in the header files. You can create it with `QTNewAtomContainer` (page 1451) and insert it with `QTInsertChild` (page 1446), using the following parameter:

Fields

`atomType`

Discussion

Value is `'beha'`.

Optional Child Atoms

`'imag'` (page 2400)
A sprite image atom.
`'crsr'` (page 2382)
Color custom cursor child atom.
`'sstr'` (page 2429)
Specifies the ID of a string variable, contained in a sprite track, to display in the status area of the browser.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'chap' Chapter or scene list track reference type atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields

`atomType`

Discussion

Constant `kTrackReferenceChapterList`, designating atom type `'chap'`.

`data`

Discussion

A list of track ID values (32-bit integers) specifying the related tracks. Note that a track ID value can be set to 0 to indicate an unused entry in the atom. Doing this can be more convenient than deleting the reference.

Parent Atom

`'tref'` (page 2442)
Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'clip' Defines a clipping region.

```
struct ClippingAtom {
    long        size;
    long        atomType;
    RgnAtom     aRgnClip;
};
```

Fields

`size`

Discussion

The size in bytes of this atom structure.

`atomType`

Discussion
Constant `ClipAID`, designating atom type `'clip'`.

`aRgnClip`

Discussion
A `'crgn'` (page 2382) atom that defines the clipping region.

Discussion
You can treat this atom either as a declared structure or as a QT atom, which you can create it with `QTInsertChild` (page 1446).

Programming Info
C interface file: `MoviesFormat.h`

See Also

For the atoms that may contain this atom, see `'moov'` (page 2414) and `'trak'` (page 2441). For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'clon' Contains information about a track clone.

```
struct CloneAtom {
    long            size;
    long            atomType;
    CloneRecord     cloneInfo;
};
```

Fields
`size`

Discussion
The size in bytes of this atom structure.

`atomType`

Discussion
Value is `'clon'`.

`cloneInfo`

Discussion
A `CloneRecord` structure.

See Also

See the `CloneRecord` structure and `AddClonedTrackToMovie` (page 1534). For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'cmov' Contains a compressed movie.

This is a QT atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameter:

Fields
`atomType`

Discussion
Constant `CompressedMovieAID`, designating atom type `'cmov'`.

Parent Atom

`'moov'` (page 2414)
Parent atom can contain only one atom of this type.

Required Child Atoms

`'dcom'` (page 2385)
Indicates the compression algorithm used to compress a movie. Only one allowed.
`'cmvd'` (page 2375)
Stores the data for a compressed movie. Only one allowed.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'cmvd' Stores the data for a compressed movie.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields
`atomType`

Discussion
Constant `CompressedMovieDataAID`, designating atom type `'cmvd'`.

`data`

Discussion
An integer of type UInt32 that gives the length of the uncompressed movie in bytes, followed by the compressed movie data.

Parent Atom

`'cmov'` (page 2374)
Parent atom can contain only one atom of this type.
'co64' A 64-bit version of the `'stco'` (page 2431) atom.

For details, see `'stco'` (page 2431).

Fields
`atomType`

Discussion
Constant STChunkOffset64AID, designating atom type 'co64'.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'©cpy' User data list entry atom: copyright information.

```
struct MoviesUserData {
    long    size;
    long    udType;
    char    data[1];
};
```

Fields
`size`

Discussion
The size in bytes of this atom structure.

`udType`

Discussion
Constant `kUserDataTextCopyright`, designating atom type `'©cpy'`.

`data`

Discussion
A string containing copyright information.

Parent Atom

`'udta'` (page 2446)

Parent atom can contain only one atom of this type.

Programming Info
C interface file: `MoviesFormat.h`

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'©day' User data list entry atom: creation date.

```
struct MoviesUserData {
    long    size;
    long    udType;
    char    data[1];
};
```

Fields
`size`

Discussion
The size in bytes of this atom structure.

`udType`

Discussion
Constant `kUserDataTextCreationDate`, designating atom type `'©day'`.

`data`

Discussion
A string containing the creation date.

Parent Atom

`'udta'` (page 2446)

Parent atom can contain only one atom of this type.

Programming Info
C interface file: `MoviesFormat.h`

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'©dir' User data list entry atom: name of movie's director.

```
struct MoviesUserData {
    long    size;
    long    udType;
    char    data[1];
};
```

Fields

`size`

Discussion

The size in bytes of this atom structure.

`udType`

Discussion

Constant `kUserDataTextDirector`, designating atom type `'©dir'`.

`data`

Discussion

A string containing the name of the movie's director.

Parent Atom

`'udta'` (page 2446)

Parent atom can contain only one atom of this type.

Programming Info

C interface file: `MoviesFormat.h`

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'©ed1' User data list entry atom: edit date 1.

```
struct MoviesUserData {
    long    size;
    long    udType;
    char    data[1];
};
```

Fields

`size`

Discussion

The size in bytes of this atom structure.

`udType`

Discussion

Constant kUserDataTextEditDate1, designating atom type '©ed1'.

`data`

Discussion

A string containing the first edit date.

Parent Atom

`'udta'` (page 2446)

Parent atom can contain only one atom of this type.

Discussion

Similar atoms of types '©ed2' through '©ed9' may contain other edit date strings.

Programming Info
C interface file: `MoviesFormat.h`

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'©fmt' User data list entry atom: indication of movie's format.

```
struct MoviesUserData {
    long    size;
    long    udType;
    char    data[1];
};
```

Fields
`size`

Discussion
The size in bytes of this atom structure.

`udType`

Discussion
Constant `kUserDataTextOriginalFormat`, designating atom type `'©fmt'`.

`data`

Discussion
A string indicating the movie's format.

Parent Atom

`'udta'` (page 2446)
Parent atom can contain only one atom of this type.

Programming Info
C interface file: `MoviesFormat.h`

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'©inf' User data list entry atom: information about the movie.

```
struct MoviesUserData {
    long    size;
    long    udType;
    char    data[1];
};
```

Fields
`size`

Discussion
The size in bytes of this atom structure.

`udType`

Discussion
Constant `kUserDataTextInformation`, designating atom type `'©inf'`.

`data`

Discussion
A string containing information about the movie.

Parent Atom

'udta' (page 2446)
Parent atom can contain only one atom of this type.

Programming Info
C interface file: `MoviesFormat.h`

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'©prd' User data list entry atom: name of movie's producer.

```
struct MoviesUserData {
    long    size;
    long    udType;
    char    data[1];
};
```

Fields
`size`

Discussion
The size in bytes of this atom structure.

`udType`

Discussion
Constant `kUserDataTextProducer`, designating atom type `'©prd'`.

`data`

Discussion
A string containing the name of the movie's producer.

Parent Atom

'udta' (page 2446)
Parent atom can contain only one atom of this type.

Programming Info
C interface file: `MoviesFormat.h`

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'©prf' User data list entry atom: names of performers.

```
struct MoviesUserData {
    long    size;
    long    udType;
    char    data[1];
};
```

Fields
`size`

Discussion
The size in bytes of this atom structure.

`udType`

Discussion
Constant `kUserDataTextPerformers`, designating atom type `'©prf'`.

`data`

Discussion
A string containing names of the performers.

Parent Atom

`'udta'` (page 2446)
Parent atom can contain only one atom of this type.

Programming Info
C interface file: `MoviesFormat.h`

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'©req' User data list entry atom: special hardware or software requirements.

```
struct MoviesUserData {
    long    size;
    long    udType;
    char    data[1];
};
```

Fields
`size`

Discussion
The size in bytes of this atom structure.

`udType`

Discussion
Constant `kUserDataTextSpecialPlaybackRequirements`, designating atom type `'©req'`.

`data`

Discussion
A string detailing special hardware or software requirements.

Parent Atom

`'udta'` (page 2446)
Parent atom can contain only one atom of this type.

Programming Info
C interface file: `MoviesFormat.h`

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'©src' User data list entry atom: credits for those who provided movie source content.

```
struct MoviesUserData {
    long    size;
    long    udType;
    char    data[1];
};
```

Fields

`size`

Discussion

The size in bytes of this atom structure.

`udType`

Discussion

Constant `kUserDataTextOriginalSource`, designating atom type `'©src'`.

`data`

Discussion

A string containing credits for those who provided movie source content.

Parent Atom

`'udta'` (page 2446)

Parent atom can contain only one atom of this type.

Programming Info

C interface file: `MoviesFormat.h`

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'©wrt' User data list entry atom: name of movie's writer.

```
struct MoviesUserData {
    long    size;
    long    udType;
    char    data[1];
};
```

Fields

`size`

Discussion

The size in bytes of this atom structure.

`udType`

Discussion

Constant `kUserDataTextWriter`, designating atom type `'©wrt'`.

`data`

Discussion

A string containing the name of the movie's writer.

Parent Atom

`'udta'` (page 2446)

Parent atom can contain only one atom of this type.

Programming Info

C interface file: `MoviesFormat.h`

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'crgn' Defines a clipping region.

```
struct RgnAtom {
    long     size;
    long     atomType;
    short    rgnSize;
    Rect     rgnBBox;
    char     data[1];
};
```

Fields
`size`

Discussion
The size in bytes of this atom structure.

`atomType`

Discussion
Constant `RgnClipAID`, designating atom type `'crgn'`.

`rgnSize`

Discussion
The size in bytes of the region.

`rgnBBox`

Discussion
The bounding box for the region.

`data`

Discussion
Additional data if the clipping region is not rectangular.

Parent Atom

`'clip'` (page 2373)
Parent atom can contain only one atom of this type.

Programming Info
C interface file: `MoviesFormat.h`

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'crsr' Color custom cursor child atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields
`atomType`

Discussion
Constant `kSpriteCursorBehaviorAtomType`, designating atom type `'crsr'`.

`data`

Discussion
A cursor description.

Parent Atom

`'vrcp'` (page 2450)

Parent atom can contain multiple atoms of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'cspd' Contains the connection speed currently set in the QuickTime preferences.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields

`atomType`

Discussion
Constant `ConnectionSpeedPrefsType`, designating atom type `'cspd'`.

`data`

Discussion
The connection speed.

See Also

See the `GetQuickTimePreference` (page 1368) and `SetQuickTimePreference` (page 1491) functions. For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'ctab' Color table atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields

`atomType`

Discussion
Constant `ColorTableAID`, designating atom type `'ctab'`.

`data`

Discussion
A color table.

Parent Atom

`'moov'` (page 2414)

Parent atom can contain only one atom of this type.

Discussion
Color table atoms define a list of preferred colors for displaying the movie on devices that support only 256 colors. The list may contain up to 256 colors.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'cufa' Non-standard cubic QTVR panorama data atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields

`atomType`

Discussion

Value is `'cufa'`.

`data`

Discussion

A `QTVRCubicFaceData` structure.

Discussion

Each entry in the `QTVRCubicFaceData` structure describes one face of the polyhedron being described.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'CURS' Custom cursor child atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields

`atomType`

Discussion

Constant `kQTVRCursorAtomType`, designating atom type `'CURS'`.

`data`

Discussion

A cursor description.

Parent Atom

`'vrcp'` (page 2450)

Parent atom can contain multiple atoms of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'cuvw' Cubic view atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields

`atomType`

Discussion

Value is `'cuvw'`.

`data`

Discussion

A `QTVRCubicViewAtom` structure.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'dasz' Data size atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields

`atomType`

Discussion

Constant `kQTTargetDataSize`, designating atom type `'dasz'`.

`data`

Discussion

A `QTTargetDataSize` structure.

Parent Atom

`'vide'` (page 2448)

Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'dcom' Indicates the compression algorithm used to compress a movie.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields

`atomType`

Discussion

Constant `DataCompressionAtomAID`, designating atom type `'dcom'`.

`data`

Discussion

A 32-bit constant (see below) that indicates which lossless algorithm was used to compress the movie contained in the parent atom.

Data Constants

`AppleDataCompressorSubType`

The Apple data compressor; value is `'adec'`.

`zlibDataCompressorSubType`

The zlib data compressor; value is `'zlib'`.

Parent Atom

`'cmov'` (page 2374)

Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'defi' A sprite image data reference atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields
`atomType`

Discussion
Constant `kSpriteImageDefaultImageIndexAtomType`, designating atom type `'defi'`.

`data`

Discussion
The image index of a traditional image, of type `short`, to use while waiting for the referenced image to load.

Parent Atom
`'imag'` (page 2400)
Parent atom can contain only one atom of this type.

Discussion
You use the this atom type to specify that an image is referenced and how to access it. Its ID should be 1.

Version Notes
Added to QuickTime 4.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'desc' Graphics export description atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields
`atomType`

Discussion
Constant `kCustomHandlerDesc`, designating atom type `'desc'`.

`data`

Discussion
A nonterminated string containing a human-readable format name.

Parent Atom
`'expo'` (page 2392)
Parent atom can contain only one atom of this type.

See Also

See the `GraphicsExportGetMIMETypeList` (page 995) and `GraphicsImportGetExportImageTypeList` (page 1047) functions. For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'dflt' Key frame shared-data atom.

This is a QT atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameter:

Fields
`atomType`

Discussion
Constant `kSpriteSharedDataAtomType`, designating atom type `'dflt'`.

Required Child Atoms

`'imct'` (page 2401)

Only one allowed.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'dimm' Number of bytes of immediate data to be sent.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields

`atomType`

Discussion

Value is `'dimm'`.

`data`

Discussion

8-byte value.

Parent Atom

`'hinf'` (page 2396)

Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'dinf' Specifies where media data is stored.

```
struct DataInfoAtom {
    long            size;
    long            atomType;
    DataRefAtom     dataRef;
};
```

Fields

`size`

Discussion

The size in bytes of this atom structure.

`atomType`

Discussion

Constant `DataInfoAID`, designating atom type `'dinf'`.

`dataRef`

Discussion

A value that contains the data for this atom. The 4-byte `DataRefAtom` data type is private and is not documented.

Optional Child Atoms

`'dref'` (page 2388)

Only one allowed.

Programming Info

C interface file: `MoviesFormat.h`

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'dmax' The largest packet duration.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields
`atomType`

Discussion
Value is `'dmax'`.

`data`

Discussion
4 bytes packet duration, in milliseconds.

Parent Atom

`'hinf'` (page 2396)
Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'dmed' Number of bytes from the media track to be sent.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields
`atomType`

Discussion
Value is `'dmed'`.

`data`

Discussion
8-byte value.

Parent Atom

`'hinf'` (page 2396)
Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'dref' Data reference atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields
`atomType`

Discussion
Constant `DataRefAID`, designating atom type `'dref'`.

`data`

Discussion
Data references.

Parent Atom

`'dinf'` (page 2387)

Parent atom can contain only one atom of this type.

See Also

See the `AliasRecord` structure. For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'drep' Number of bytes of repeated data to be sent.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields

`atomType`

Discussion
Value is `'drep'`.

`data`

Discussion
8-byte value.

Parent Atom

`'hinf'` (page 2396)

Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'edts' Contains an atom that defines an edit list.

```
struct EditsAtom {
    long            size;
    long            atomType;
    EditListAtom    editList;
};
```

Fields

`size`

Discussion
The size in bytes of this atom structure.

`atomType`

Discussion
Constant `EditsAID`, designating atom type `'edts'`.

`editList`

Discussion
An `'elst'` (page 2390) atom.

Parent Atom

'trak' (page 2441)

Parent atom can contain only one atom of this type.

Required Child Atoms

'elst' (page 2390)

Only one allowed.

Programming Info

C interface file: `MoviesFormat.h`

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'elst' Contains a list of edit segment definitions for a media.

```
struct EditListAtom {
    long            size;
    long            atomType;
    long            flags;
    long            numEntries;
    EditListType    editListTable[1];
};
```

Fields

`size`

Discussion

The size in bytes of this atom structure.

`atomType`

Discussion

Constant `EditListAID`, designating atom type `'elst'`.

`flags`

Discussion

One byte of version information followed by three bytes of flags. The flag bytes are not currently used.

`numEntries`

Discussion

The number of entries in `editListTable`.

`editListTable`

Discussion

An array of `EditListType` data structures, each of which locates and defines an edit segment within a media.

Parent Atom

'edts' (page 2389)

Parent atom can contain only one atom of this type.

Discussion

You can use the edit list atom to tell QuickTime how to map from a time in a movie to a time in a media, and ultimately to each segment of the media's data.

Programming Info

C interface file: `MoviesFormat.h`

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'end ' Defines the ending offset of hypertext in a text stream.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields

`atomType`

Discussion

Value is `'end '` (the fourth character is a space).

`data`

Discussion

The ending offset of hypertext in a text stream.

Parent Atom

`'htxt'` (page 2399)

Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'enda' Determines the endian status of the sound component that interprets data contained in an audio atom list.

```
struct AudioEndianAtom {
    long      size;
    OSType    atomType;
    short     littleEndian;
};
```

Fields

`size`

Discussion

The size in bytes of this atom structure.

`atomType`

Discussion

Constant `kAudioEndianAtomType`, designating atom type `'enda'`.

`littleEndian`

Discussion

Set this field to TRUE if the audio component is to operate on little-endian data, and FALSE otherwise.

Programming Info

C interface file: `Sound.h`

See Also

To choose the sound component for an audio atom list, see the `'frma'` (page 2393) atom. For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'expo' Defines a graphics export group.

This is a QT container atom; it is not declared in the header files. You can create it with `QTNewAtomContainer` (page 1451) and insert it with `QTInsertChild` (page 1446), using the following parameters:

Fields
`atomType`

Discussion
Constant `kGraphicsExportGroup`, designating atom type `'expo'`.

Required Child Atoms

`'ftyp'` (page 2394)
An `OSType` representing the exported file type.
`'ext '` (page 2392)
A nonterminated string containing the suggested file extension for this format.
`'desc'` (page 2386)
A nonterminated string containing a human-readable name for this format.

Optional Child Atoms

`'mime'[atom]` (page 2410)
A nonterminated string containing the MIME type for this format.

See Also

See the `GraphicsImportGetExportImageTypeList` (page 1047) function. For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'ext ' Defines a graphics export extension.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields
`atomType`

Discussion
Constant `kGraphicsExportExtension`, designating atom type `'ext '` (the fourth character is a space).

`data`

Discussion
A nonterminated string containing a file extension.

Parent Atom

`'expo'` (page 2392)
Parent atom can contain only one atom of this type.

See Also

See the `GraphicsImportGetExportImageTypeList` (page 1047) function. For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'flap' Extension to the `SoundDescription` structure.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields
`atomType`

Discussion
Constant `siSlopeAndIntercept`, designating atom type `'flap'`; see `Sound Information Selectors`.

`data`

Discussion
A `SoundSlopeAndInterceptRecord` structure.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'flov' Contains a floating-point variable for a sprite.

This is a QT container atom; it is not declared in the header files. You can create it with `QTNewAtomContainer` (page 1451) and insert it with `QTInsertChild` (page 1446), using the following parameter:

Fields
`atomType`

Discussion
Value is `'flov'`.

Parent Atom

`'vars'` (page 2448)
Contains variables for a sprite.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'free' Provides unused space in a movie file.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields
`atomType`

Discussion
Constant `FreeAtomType`, designating atom type `'free'`.

`data`

Discussion
Any number of bytes of free space.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'frma' Specifies which sound component is responsible for the atoms contained in an audio atom list.

```
struct AudioFormatAtom {
    long      size;
    OSType    atomType;
    OSType    format;
};
```

Fields
`size`

Discussion
The size in bytes of this atom structure.

`atomType`

Discussion
Constant `kAudioFormatAtomType`, designating atom type `'frma'`.

`format`

Discussion
A constant that identifies a sound component. See `Codec Identifiers`.

Programming Info
C interface file: `Sound.h`

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'ftyp' Defines a graphics export file type.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields
`atomType`

Discussion
Constant `kGraphicsExportFileType`, designating atom type `'ftyp'`.

`data`

Discussion
An `OSType` representing the exported file type.

Parent Atom

`'expo'` (page 2392)
Parent atom can contain only one atom of this type.

See Also

See the `GraphicsImportGetExportImageTypeList` (page 1047) function. For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'gmhd' Contains a generic media information atom.

This is a QT container atom; it is not declared in the header files. You can create it with `QTNewAtomContainer` (page 1451) and insert it with `QTInsertChild` (page 1446), using the following parameter:

Fields
`atomType`

Discussion
Constant `GenericMediaInfoHeaderAID`, designating atom type `'gmhd'`.

Parent Atom

`'minf'[generic]` (page 2411)
Parent atom can contain only one atom of this type.

Required Child Atoms

`'gmin'` (page 2395)
Provides data that is specific to a handler for media other than video or sound. Only one allowed.

Discussion
This atom is currently used only as a container for a `'gmin'` (page 2395) atom.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'gmin' Provides data that is specific to a handler for media other than video or sound.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields

`atomType`

Discussion
Constant `GenericMediaInfoAID`, designating atom type `'gmin'`.

`data`

Discussion
Data required by the media handler that is designated by the `'hdlr'` (page 2395) atom contained in the `'minf'[generic]` (page 2411) atom that also contains the parent of this atom.

Parent Atom

`'gmhd'` (page 2394)
Parent atom can contain any number of atoms of this type.

Discussion
This atom contains handler-specific information to support your use of a `'minf'[generic]` (page 2411) atom. Note that the data in this atom is not used by RTP servers.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'hdlr' Specifies the component that is to interpret a media's data.

```
struct HandlerAtom {
    long                    size;
    long                    atomType;
    PublicHandlerInfo       hInfo;
};
```

Fields

`size`

Discussion
The size in bytes of this atom structure.

`atomType`

Discussion
Constant `HandlerAID`, designating atom type `'hdlr'`.

`hInfo`

Discussion
A `PublicHandlerInfo` structure, which contains the actual data for this atom.

Discussion
RTP servers ignore this atom's data when it is contained in a `'minf'[generic]` (page 2411) atom.

Programming Info

C interface file: `MoviesFormat.h`

See Also

For the atoms that may contain this atom, see 'mdia' (page 2409), 'minf'[generic] (page 2411), 'minf'[sound] (page 2412), and 'minf'[video] (page 2413). For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'hinf' Contains statistics for the hint track.

This is a QT container atom; it is not declared in the header files. You can create it with QTNewAtomContainer (page 1451) and insert it with QTInsertChild (page 1446), using the following parameter:

Fields
atomType

Discussion
Value is 'hinf'.

Required Child Atoms

'trpy' (page 2443)
Total number of bytes that will be sent, including 12-byte RTP headers but not including network headers. Only one allowed.
'nump' (page 2417)
Total number of network packets that will be sent. Only one allowed.
'tpyl' (page 2440)
Total number of bytes that will be sent, not including 12-byte RTP headers. Only one allowed.
'maxr' (page 2408)
Maximum data rate. Only one allowed.
'dmed' (page 2388)
Number of bytes from the media track to be sent. Only one allowed.
'dimm' (page 2387)
Number of bytes of immediate data to be sent. Only one allowed.
'drep' (page 2389)
Number of bytes of repeated data to be sent. Only one allowed.
'tmin' (page 2440)
Smallest relative transmission time, in milliseconds. Only one allowed.
'tmax' (page 2439)
Largest relative transmission time, in milliseconds. Only one allowed.
'pmax' (page 2419)
Largest packet, in bytes, including 12-byte RTP header. Only one allowed.
'dmax' (page 2388)
The largest packet duration. Only one allowed.
'payt' (page 2418)
Payload type. Only one allowed.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'hint' Hint track reference type atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with QTInsertChild (page 1446) using the following parameters:

Fields
atomType

Discussion
Value is 'hint'.

`data`

Discussion
A list of track ID values (32-bit integers) specifying the related tracks. Note that a track ID value can be set to 0 to indicate an unused entry in the atom. Doing this can be more convenient than deleting the reference.

Parent Atom

`'tref'` (page 2442)
Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'hlit' Defines the highlighted portion in text.

```
struct HiliteAtom {
    long    size;
    long    atomType;
    long    selStart;
    long    selEnd;
};
```

Fields
`size`

Discussion
The size in bytes of this atom structure.

`atomType`

Discussion
Value is `'hlit'`.

`selStart`

Discussion
Character number of highlighted selection start character.

`selEnd`

Discussion
Character number of highlighted selection end character.

Discussion
For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

Programming Info
C interface file: `MoviesFormat.h`

'hnti' Hint track user data atom.

This is a QT container atom; it is not declared in the header files. You can create it with `QTNewAtomContainer` (page 1451) and insert it with `QTInsertChild` (page 1446), using the following parameter:

Fields
`atomType`

Discussion
Value is `'hnti'`.

Required Child Atoms

`'sdp '` (page 2426)
SDP text for a hint track. Only one allowed.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'hots' A QTVR hot spot.

This is a QT container atom; it is not declared in the header files. You can create it with `QTNewAtomContainer` (page 1451) and insert it with `QTInsertChild` (page 1446), using the following parameter:

Fields
`atomType`

Discussion
Constant `kQTVRHotSpotAtomType`, designating atom type `'hots'`.

Parent Atom

`'hspa'` (page 2398)
Parent atom can contain any number of atoms of this type.

Required Child Atoms

`'hsin'` (page 2398)
Contains general hot spot information. Only one allowed.
`'link'` (page 2406)
Specific information about a link hot spot. Only one allowed.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'hsin' Contains general hot spot information.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields
`atomType`

Discussion
Constant `kQTVRHotSpotInfoAtomType`, designating atom type `'hsin'`.

`data`

Discussion
Hot spot information.

Parent Atom

`'hots'` (page 2398)
Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'hspa' Hot spot parent atom.

This is a QT container atom; it is not declared in the header files. You can create it with `QTNewAtomContainer` (page 1451) and insert it with `QTInsertChild` (page 1446), using the following parameter:

Fields
`atomType`

Discussion
Constant `kQTVRHotSpotParentAtomType`, designating atom type `'hspa'`.

Parent Atom

`'vrnp'` (page 2450)
Parent atom can contain only one atom of this type.

Required Child Atoms

`'hots'` (page 2398)
A QTVR hot spot. Any number allowed.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'htxt' Hypertext in a text stream.

This is a QT container atom; it is not declared in the header files. You can create it with `QTNewAtomContainer` (page 1451) and insert it with `QTInsertChild` (page 1446), using the following parameter:

Fields
`atomType`

Discussion
Value is `'htxt'`.

Parent Atom

`'wtxt'` (page 2453)
Parent atom can contain multiple atoms of this type.

Required Child Atoms

`'strt'` (page 2432)
Defines the starting offset of hypertext in a text stream. Only one allowed.
`'end '` (page 2391)
Defines the ending offset of hypertext in a text stream. Only one allowed.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'idat' Image data atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields
`atomType`

Discussion
Constant `quickTimeImageFileImageDataAtom`, designating atom type `'idat'`.

`data`

Discussion
The image data.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'idsc' Image description atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields

`atomType`

Discussion

Constant `quickTimeImageFileImageDescriptionAtom`, designating atom type `'idsc'`.

`data`

Discussion

The image description.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'iicc' ColorSync profile atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields

`atomType`

Discussion

Constant `quickTimeImageFileColorSyncProfileAtom`, designating atom type `'iicc'`.

`data`

Discussion

A ColorSync profile.

Version Notes

This is a new optional atom in QuickTime 4.

See Also

See the `GraphicsExportSetColorSyncProfile` (page 1006) function. For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'imag' A sprite image atom.

This is a QT container atom; it is not declared in the header files. You can create it with `QTNewAtomContainer` (page 1451) and insert it with `QTInsertChild` (page 1446), using the following parameter:

Fields

`atomType`

Discussion

Constant `kSpriteImageAtomType`, designating atom type `'imag'`.

Parent Atom

`'imct'` (page 2401)

Parent atom can contain any number of atoms of this type.

Required Child Atoms

`'imda'` (page 2401)

Contains sprite image data. Only one allowed.

Optional Child Atoms

`'name'[sprite]` (page 2416)
A sprite name atom. Only one allowed.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'imap' An input map.

This is a QT container atom; it is not declared in the header files. You can create it with `QTNewAtomContainer` (page 1451) and insert it with `QTInsertChild` (page 1446), using the following parameter:

Fields
`atomType`

Discussion
Constant `InputMapAID`, designating atom type `'imap'`.

Parent Atom

`'trak'` (page 2441)
Parent atom can contain only one atom of this type.

Required Child Atoms

`' in'` (page 2405)
`Track` input atom. Only one allowed.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'imct' A sprite image container atom.

This is a QT container atom; it is not declared in the header files. You can create it with `QTNewAtomContainer` (page 1451) and insert it with `QTInsertChild` (page 1446), using the following parameter:

Fields
`atomType`

Discussion
Constant `kSpriteImagesContainerAtomType`, designating atom type `'imct'`.

Parent Atom

`'dflt'` (page 2386)
Parent atom can contain only one atom of this type.

Required Child Atoms

`'imag'` (page 2400)
`Sprite` image atom. Any number allowed.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'imda' A sprite image data.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields
`atomType`

Discussion
Constant `kSpriteImageDataAtomType`, designating atom type `'imda'`.

`data`

Discussion
Image data.

Parent Atom

`'imag'` (page 2400)
Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'imgp' Panorama imaging parent atom.

This is a QT container atom; it is not declared in the header files. You can create it with `QTNewAtomContainer` (page 1451) and insert it with `QTInsertChild` (page 1446), using the following parameter:

Fields
`atomType`

Discussion
Constant `kQTVRImagingParentAtomType`, designating atom type `'imgp'`.

Required Child Atoms

`'impn'` (page 2403)
Panorama imaging atom. Any number allowed.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'imgr' A sprite image group ID atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields
`atomType`

Discussion
Constant `kSpriteImageGroupIDAtomType`, designating atom type `'imgr'`.

`data`

Discussion
The group ID, of type long.

Parent Atom

`'imag'` (page 2400)
Parent atom can contain only one atom of this type.

Discussion
Each image in a sprite media key frame sample is assigned to a group. Add an atom of this type as a child of the `'imag'` (page 2400) atom and set its leaf data to a long containing the group ID. For example, if the sample contains ten images where the first two images are equivalent, and the last eight images are equivalent, then you could assign a group ID of 1000 to the first two images, and a group ID of 1001 to the last eight images. This divides the images in the sample into two sets. The actual ID does not matter, it just needs to be a unique positive integer. Note that you must assign group IDs to your sprite sample if you want a sprite to display images with non-equivalent image descriptions (i.e., images with different dimensions).

Special Considerations

Although QuickTime does not currently use this atom internally, tools that edit sprite media can use the information provided to optimize certain operations, such as cut, copy, and paste.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'impn' Panorama imaging atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields

`atomType`

Discussion
Constant `kQTVRPanoImagingAtomType`, designating atom type `'impn'`.

`data`

Discussion
A `QTVRPanoImagingAtom` structure.

Parent Atom

`'imgp'` (page 2402)
Parent atom can contain any number of atoms of this type.

Discussion
A `QTVRPanoImagingAtom` describes the default imaging characteristics for all the panoramic nodes in a scene. This atom overrides QuickTime VR's own defaults.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'imre' A sprite image data reference atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields

`atomType`

Discussion
Constant `kSpriteImageDataRefAtomType`, designating atom type `'imre'`.

`data`

Discussion
The data reference, which is similar to the `dataRef` parameter of `GetDataHandler` (page 1569).

Parent Atom

`'imag'` (page 2400)
Parent atom can contain only one atom of this type.

Discussion
You use the this atom type to specify that an image is referenced and how to access it. Add this atom as a child of the `'imag'` (page 2400) atom instead of an `'imda'` (page 2401) atom. Its ID should be 1.

Version Notes
Added in QuickTime 4.

See Also

See the `GetDataHandler` (page 1569) function. For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'imrg' Custom sprite image registration point atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields

`atomType`

Discussion

Constant `kSpriteImageRegistrationAtomType`, designating atom type `'imrg'`.

`data`

Discussion

The desired sprite registration point, a `FixedPoint` structure.

Parent Atom

`'imag'` (page 2400)

Parent atom can contain only one atom of this type.

Discussion

`Sprite` images have a default registration point of 0, 0. To specify a different point, add an atom of this type as a child atom of the `'imag'` (page 2400) and set its leaf data to a `FixedPoint` value with the desired registration point.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'imrt' A sprite image data reference type atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields

`atomType`

Discussion

Constant `kSpriteImageDataRefTypeAtomType`, designating atom type `'imrt'`.

`data`

Discussion

The data reference type, which is similar to the `dataRefType` parameter of `GetDataHandler` (page 1569).

Parent Atom

`'imag'` (page 2400)

Parent atom can contain only one atom of this type.

Discussion

You use the this atom type to specify that an image is referenced and how to access it. Add this atom as a child of the `'imag'` (page 2400) atom. Its ID should be 1.

Version Notes

Added in QuickTime 4.

See Also

See the GetDataHandler (page 1569) function. For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

' in' Track input atom.

This is a QT container atom; it is not declared in the header files. You can create it with QTNewAtomContainer (page 1451) and insert it with QTInsertChild (page 1446), using the following parameter:

Fields
atomType

Discussion
Value ' in'; the first two characters are spaces.

Parent Atom

'imap' (page 2401)
Parent atom can contain only one atom of this type.

Required Child Atoms

' ty' (page 2445)
Input atom type. Only one allowed.

Optional Child Atoms

'obid' (page 2418)
Object ID atom. Only one allowed.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'kmat' Defines a matte for a track's compressed media.

```
struct MatteCompressedAtom {
    long                long size;
    long                atomType;
    long                flags;
    ImageDescription    matteImageDescription;
    char                matteData[1];
};
```

Fields
size

Discussion
The size in bytes of this atom structure.

atomType

Discussion
Constant MatteCompAID, designating atom type 'kmat'.

flags

Discussion
One byte of version information followed by three bytes of flags. The flags bytes are not currently used.

matteImageDescription

Discussion
An ImageDescription data structure for the matte.

`matteData`

Discussion
An array of matte data.

Programming Info
C interface file: `MoviesFormat.h`

See Also

For the structure that contains this atom, see the `'matt'` (page 2407) atom. For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'link' Contains specific information about a link hot spot.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields
`atomType`

Discussion
Constant `kQTVRLinkInfoAtomType`, designating atom type `'link'`.

`data`

Discussion
Link hot spot information.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'load' Contains preloading information for a track.

```
struct TrackLoadSettingsAtom {
    long                    size;
    long                    atomType;
    TrackLoadSettings       settings;
};
```

Fields
`size`

Discussion
The size in bytes of this atom structure.

`atomType`

Discussion
Constant `LoadSettingsAID`, designating atom type `'load'`.

`settings`

Discussion
A `TrackLoadSettings` data structure, which contains the actual data for this atom.

Programming Info
C interface file: `MoviesFormat.h`

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'LOOP' User data list entry atom: looping `style`.

```
struct MoviesUserData {
    long    size;
    long    udType;
    char    data[1];
};
```

Fields

`size`

Discussion

The size in bytes of this atom structure.

`udType`

Discussion

Value is `'LOOP'`.

`data`

Discussion

A long integer, indicating looping `style`: 0 for normal looping, 1 for palindromic looping.

Parent Atom

`'udta'` (page 2446)

Parent atom can contain only one atom of this type.

Discussion

This atom is present only if the movie is set to loop.

Programming Info

C interface file: `MoviesFormat.h`

See Also

See the `MoviesUserData` structure. For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'matt' Defines a matte for a track's media.

```
struct MatteAtom {
    long                    size;
    long                    atomType;
    MatteCompressedAtom     aCompressedMatte;
};
```

Fields

`size`

Discussion

The size in bytes of this atom structure.

`atomType`

Discussion

Constant `MatteAID`, designating atom type `'matt'`.

`aCompressedMatte`

Discussion

A `'kmat'` (page 2405) atom.

Required Child Atoms

`'kmat'` (page 2405)
Only one allowed.

Programming Info
C interface file: `MoviesFormat.h`

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'maxr' Maximum data rate.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields
`atomType`
Discussion
Value is `'maxr'`.

`data`
Discussion
8 bytes.

Parent Atom

`'hinf'` (page 2396)
Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'mdat' Media data atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields
`atomType`
Discussion
Constant `MovieDataAtomType`, designating atom type `'mdat'`.

`data`
Discussion
Media data.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'mdhd' Specifies the characteristics of a media.

```
struct MediaHeaderAtom {
    long            size;
    long            atomType;
    MediaHeader     header;
};
```

Fields
`size`

Discussion
The size in bytes of this atom structure.

`atomType`

Discussion
Constant `MediaHeaderAID`, designating atom type `'mdhd'`.

`header`

Discussion
A `MediaHeader` data structure, which contains the actual data for this atom.

Parent Atom

`'mdia'` (page 2409)
Parent atom can contain only one atom of this type.

Programming Info
C interface file: `MoviesFormat.h`

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'mdia' Defines the media for a movie track.

```
struct MediaDirectory {
    long                size;
    long                atomType;
    MediaHeaderAtom     mediaHeader;
    HandlerAtom         mediaHandler;
    MediaInfo           mediaInfo;
};
```

Fields
`size`

Discussion
The size in bytes of this atom structure.

`atomType`

Discussion
Constant `MediaAID`, designating atom type `'mdia'`.

`mediaHeader`

Discussion
A `'mdhd'` (page 2409) atom that specifies general characteristics of the media.

`mediaHandler`

Discussion
A `'hdlr'` (page 2395) atom that defines a handler for the media.

`mediaInfo`

Discussion

A `'minf'[generic]` (page 2411) atom structure that contains data to be passed to the media handler.

Parent Atom

`'trak'` (page 2441)
Parent atom can contain only one atom of this type.

Required Child Atoms

`'mdhd'` (page 2409)
General characteristics of the media. Only one allowed.

Optional Child Atoms

`'hdlr'` (page 2395)
The type of media this atom contains. Only one allowed.
`'minf'[generic]` (page 2411)
Data that is specific to a media handler. Only one allowed.
`'udta'` (page 2446)
User data atom. Only one allowed.

Discussion

The `'hdlr'` atom specifies what type of media this atom contains; for example, video or sound. The content of the `'minf'[generic]` atom is specific to the media handler that is to interpret the media.

Programming Info

C interface file: `MoviesFormat.h`

See Also

See the `MediaHeader` structure. For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'mime'[atom] Defines a graphics export MIME type.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields

`atomType`

Discussion

Constant `kGraphicsExportMIMEType`, designating atom type `'mime'`.

`data`

Discussion

A nonterminated string containing a MIME type.

Parent Atom

`'expo'` (page 2392)
Parent atom can contain only one atom of this type.

See Also

See the `GraphicsImportGetExportImageTypeList` (page 1047), `GraphicsImportGetMIMETypeList` (page 1055), and `GraphicsExportGetMIMETypeList` (page 995) functions. For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'minf'[base] Provides data that is specific to a handler for media other than video or sound.

```
struct MediaInfo {
    long    size;
    long    atomType;
};
```

Fields
`size`

Discussion
The size in bytes of this atom structure.

`atomType`

Discussion
Constant `MediaInfoAID`, designating atom type `'minf'`.

Parent Atom

`'mdia'` (page 2409)
Parent atom can contain only one atom of this type.

Required Child Atoms

`'gmhd'` (page 2394)
Generic media information atom. Only one allowed.
`'gmin'` (page 2395)
Provides data that is specific to a handler for media other than video or sound. Only one allowed.

Discussion
Media information atoms store handler-specific information for the media data that constitutes a track. The media handler uses this information to map from media time to media data. The format and content of media information atoms are dictated by the media handler that is responsible for interpreting the media data stream. Another media handler would not know how to interpret this information.

Programming Info
C interface file: `MoviesFormat.h`

See Also

This isotope of the `'minf'` atom provides data that is specific to a handler for media other than video or sound. Handler-specific data for sound and video are provided by the `'minf'[sound]` (page 2412) atom and the `'minf'[video]` (page 2413) atom. For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'minf'[generic] Provides data that is specific to a handler for media other than video or sound.

```
struct MediaInfo {
    long    size;
    long    atomType;
};
```

Fields
`size`

Discussion
The size in bytes of this atom structure.

`atomType`

Discussion
Constant `MediaInfoAID`, designating atom type `'minf'`.

Parent Atom

'mdia' (page 2409)
Parent atom can contain only one atom of this type.

Required Child Atoms

'gmhd' (page 2394)
Generic media information atom. Only one allowed.
'hdlr' (page 2395)
The type of media this atom contains. Only one allowed.

Optional Child Atoms

'dinf' (page 2387)
Specifies where media data is stored. Only one allowed.
'stbl' (page 2430)
Contains information for converting from media time to sample number to sample location and indicates how to interpret samples and chunks. Only one allowed.

Discussion
Media information atoms store handler-specific information for the media data that constitutes a track. The media handler uses this information to map from media time to media data. The format and content of media information atoms are dictated by the media handler that is responsible for interpreting the media data stream. Another media handler would not know how to interpret this information.

Programming Info
C interface file: `MoviesFormat.h`

See Also

This isotope of the `'minf'` atom provides data that is specific to a handler for media other than video or sound. Handler-specific data for sound and video are provided by the 'minf'[sound] (page 2412) atom and the 'minf'[video] (page 2413) atom. For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'minf'[sound] Sound media information atom.

```
struct MediaInfo {
    long    size;
    long    atomType;
};
```

Fields
`size`

Discussion
The size in bytes of this atom structure.

`atomType`

Discussion
Constant `MediaInfoAID`, designating atom type `'minf'`.

Parent Atom

'mdia' (page 2409)
Parent atom can contain only one atom of this type.

Required Child Atoms

'smhd' (page 2427)
Contains sound stereo balance information. Only one allowed.

'hdlr' (page 2395)
The type of media this atom contains. Only one allowed.

Optional Child Atoms

'dinf' (page 2387)
Specifies where media data is stored. Only one allowed.
'stbl' (page 2430)
Contains information for converting from media time to sample number to sample location and indicates how to interpret samples and chunks. Only one allowed.

Discussion
Media information atoms store handler-specific information for the media data that constitutes a track. The media handler uses this information to map from media time to media data. The format and content of media information atoms are dictated by the media handler that is responsible for interpreting the media data stream. Another media handler would not know how to interpret this information.

Programming Info
C interface file: `MoviesFormat.h`

See Also

This isotope of the `'minf'` atom provides handler-specific data for sound. Handler-specific data for video is provided by the the `'minf'[video]` (page 2413) atom. For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'minf'[video] Video media information atom.

```
struct MediaInfo {
    long    size;
    long    atomType;
};
```

Fields
`size`

Discussion
The size in bytes of this atom structure.

`atomType`

Discussion
Constant `MediaInfoAID`, designating atom type `'minf'`.

Parent Atom

'mdia' (page 2409)
Parent atom can contain only one atom of this type.

Required Child Atoms

'vmhd'[media] (page 2448)
Stores handler-specific information for video media in a track. Only one allowed.
'hdlr' (page 2395)
The type of media this atom contains. Only one allowed.

Optional Child Atoms

'dinf' (page 2387)
Specifies where media data is stored. Only one allowed.
'stbl' (page 2430)
Contains information for converting from media time to sample number to sample location and indicates how to interpret samples and chunks. Only one allowed.

Discussion
Media information atoms store handler-specific information for the media data that constitutes a track. The media handler uses this information to map from media time to media data. The format and content of media information atoms are dictated by the media handler that is responsible for interpreting the media data stream. Another media handler would not know how to interpret this information.

Programming Info
C interface file: `MoviesFormat.h`

See Also

This isotope of the `'minf'` atom provides handler-specific data for video. Handler-specific data for sound is provided by the `'minf'[sound]` (page 2412) atom. For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'moov' Contains the top-level atoms that constitute a movie.

```
struct MovieDirectory {
    long                    size;
    long                    atomType;
    MovieHeaderAtom         header;
    ClippingAtom            movieClip;
    TrackDirectoryEntry     track[1];
    UserDataAtom            userData;
};
```

Fields
`size`

Discussion
The size in bytes of this atom structure.

`atomType`

Discussion
Constant `MovieAID`, designating atom type `'moov'`.

`header`

Discussion
A `'mvhd'` (page 2415) atom that specifies the general characteristics of the movie.

`movieClip`

Discussion
A `'clip'` (page 2373) atom that defines the clipping region for the movie.

`track`

Discussion
An array of one or more `TrackDirectoryEntry` data structures, each of which includes a `'trak'` (page 2441) atom that defines a track in the movie.

`userData`

Discussion
A `'udta'` (page 2446) atom, which contains user data.

Required Child Atoms

`'mvhd'` (page 2415)
Specifies the general characteristics of a movie. Only one allowed.

Optional Child Atoms

`'clip'` (page 2373)
Defines the clipping region for the movie. Only one allowed.
`'trak'` (page 2441)
Defines a single track of the movie. Any number allowed.
`'udta'` (page 2446)
User data atom. Only one allowed.
`'ctab'` (page 2383)
Color table atom. Only one allowed.
`'ptv '` (page 2419)
Defines a movie's full screen mode.

Discussion
You use movie atoms to specify the information that defines a movie; that is, the information that allows your application to understand the data that is stored in the movie data atom. The movie atom contains the movie header atom, which defines the time scale and duration information for the entire movie, as well as its display characteristics. In addition, the movie atom contains each track in the movie.

Programming Info
C interface file: `MoviesFormat.h`

See Also

See the `MovieHeader` structure. For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'mvhd' Specifies the characteristics of an entire movie.

```
struct MovieHeaderAtom {
    long            size;
    long            atomType;
    MovieHeader     header;
};
```

Fields
`size`

Discussion
The size in bytes of this atom structure.

`atomType`

Discussion
Constant `MovieHeaderAID`, designating atom type `'mvhd'`.

`header`

Discussion
A `MovieHeader` data structure, which contains the actual data for this atom.

Parent Atom

`'moov'` (page 2414)
Parent atom can contain only one atom of this type.

Discussion
You use the movie header atom to specify the characteristics of an entire QuickTime movie. The data contained in this atom defines characteristics of the entire QuickTime movie, such as time scale and duration.

Programming Info
C interface file: `MoviesFormat.h`

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'name'[sprite] A sprite name atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields

`atomType`

Discussion

Constant `kSpriteNameAtomType`, designating atom type `'name'`.

`data`

Discussion

One or more ASCII characters comprising the sprite's name.

Parent Atom

`'sprt'` (page 2428)

Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'name'[userdata] User data list entry atom: name of object.

```
struct MoviesUserData {
    long    size;
    long    udType;
    char    data[1];
};
```

Fields

`size`

Discussion

The size in bytes of this atom structure.

`udType`

Discussion

Constant `kUserDataName`, designating atom type `'name'`.

`data`

Discussion

A name string.

Parent Atom

`'udta'` (page 2446)

Parent atom can contain only one atom of this type.

Programming Info

C interface file: `MoviesFormat.h`

See Also

See the `MoviesUserData` structure. For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'ndhd' Node header atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields
`atomType`
Discussion
Constant `kQTVRNodeHeaderAtomType`, designating atom type `'ndhd'`.

`data`
Discussion
Node header information.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'nloc' QTVR node location atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields
`atomType`
Discussion
Constant `kQTVRNodeLocationAtomType`, designating atom type `'nloc'`.

`data`
Discussion
Node location.

Parent Atom
`'vrni'` (page 2450)
Parent atom can contain only one atom of this type.
See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'nump' Total number of network packets that will be sent.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields
`atomType`
Discussion
Value is `'nump'`.

`data`
Discussion
8 bytes.

Parent Atom
`'hinf'` (page 2396)
Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'obid' Object ID atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields

`atomType`

Discussion
Constant `kTrackModifierObjectID`, designating atom type `'obid'`.

`data`

Discussion
The object ID.

Parent Atom

`' in'` (page 2405)
Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'payt' Payload type atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields

`atomType`

Discussion
Value is `'payt'`.

`data`

Discussion
Payload type, which includes payload number (32-bits) followed by an RTP map payload string (a Pascal string).

Parent Atom

`'hinf'` (page 2396)
Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'pdat' Panorama sample atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields

`atomType`

Discussion
Constant `kQTVRPanoSampleDataAtomType`, designating atom type `'pdat'`.

`data`

Discussion
A panorama sample.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'pmax' Largest packet, in bytes; includes 12-byte RTP header.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields
`atomType`

Discussion
Value is `'pmax'`.

`data`

Discussion
4 bytes.

Parent Atom

`'hinf'` (page 2396)
Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'pnot' Reference to movie preview data.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields
`atomType`

Discussion
Constant `ShowFilePreviewComponentType`, designating atom type `'pnot'`.

`data`

Discussion
Reference to a movie preview.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'ptv ' Defines a movie's full screen mode.

This is a classic atom; you can access its information by calculating offsets.

Fields
`size`

Discussion
Value is 0x0000000C.

`atomType`

Discussion
Value is `'ptv '`.

Parent Atom

`'moov'` (page 2414)
Contains the top-level atoms that constitute a movie.

Data offsets

0x0000
Display size: a 16-bit big-endian integer (see below) indicating the display size for the movie.
0x0002
Reserved: set to 0.
0x0004
Reserved: set to 0.
0x0006
Slide show: an 8-bit Boolean whose value is 1 for a slide show. In slide show mode, the movie advances one frame each time the right-arrow key is pressed. Audio is muted.
0x0007
Play on open: an 8-bit boolean whose value is normally 1, indicating that the movie should play when opened. Since there is no visible controller in full-screen mode, applications should always set this field to 1 to prevent user confusion.

Display size constants

0x00000
The movie should be played at its normal size.
0x0001
The movie should be played at double size.
0x0002
The movie should be played at half size.
0x0003
The movie should be scaled to fill the screen.
0x0004
The movie should be played at its current size. This value is normally used when the `'ptv '` atom is inserted transiently and the movie has been temporarily resized.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'qdrg' QuickDraw region atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields
`atomType`

Discussion
Constant `kTweenRegionData`, designating atom type `'qdrg'`.

`data`

Discussion
Two `Rect` structures and a `MacRegion` structure.

Discussion
This atom's ID must be 1.

See Also

See the `TweenerInitialize` (page 550) function. For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'rdrf' Provides a reference to an alternate movie.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields
`atomType`

Discussion
Constant `ReferenceMovieDataRefAID`, designating atom type `'rdrf'`.

`data`

Discussion
A `ReferenceMovieDataRefRecord` data structure. The alternate movie referenced by this structure is the movie associated with the parent `'rmda'` (page 2422) atom.

Parent Atom

`'rmda'` (page 2422)
Parent atom can contain only one atom of this type.

Discussion
Alias data references are the contents of `AliasRecord` structures. The QuickTime plug-in is smart enough to convert a relative alias to a relative URL. To designate the anchor file for a relative alias, pass the `FSSpec` structure that specifies the file you are creating. You can pass absolute or relative URLs; if the movie is loaded from the desktop, QuickTime will convert a relative URL into a relative alias.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'reso' `Pixmap` resolution atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields
`atomType`

Discussion
Constant `kQTResolutionSettings`, designating atom type `'reso'`.

`data`

Discussion
A `QTResolutionSettings` structure.

Parent Atom

`'vide'` (page 2448)
Parent atom can contain only one atom of this type.

Discussion
This atom specifies the resolution for the `PixMap` structure passed to the compressor.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'rmcd' Provides component availability information for selecting an alternate movie.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields

`atomType`

Discussion

Constant `ReferenceMovieComponentCheckAID`, designating atom type `'rmcd'`.

`data`

Discussion

A `QTAltComponentCheckRecord` data structure.

Parent Atom

`'rmda'` (page 2422)

Parent atom can contain any number of atoms of this type.

See Also

See the `QTAltComponentCheckRecord` structure. For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'rmcs' Provides CPU speed information for selecting an alternate movie.

This is a QT atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields

`atomType`

Discussion

Constant `ReferenceMovieCPURatingAID`, designating atom type `'rmcs'`.

`data`

Discussion

A `QTAltCPURatingRecord` data structure.

Parent Atom

`'rmda'` (page 2422)

Parent atom can contain only one atom of this type.

See Also

See the `QTAltCPURatingRecord` structure. For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'rmda' Provides criteria for selecting an alternate movie.

This is a QT atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameter:

Fields

`atomType`

Discussion

Constant `ReferenceMovieDescriptorAID`, designating atom type `'rmda'`.

Parent Atom

'rmra' (page 2424)
Parent atom can contain only one atom of this type.

Required Child Atoms

'rdrf' (page 2421)
A reference to an alternate movie. Only one allowed.

Optional Child Atoms

'rmdr' (page 2423)
Data rate information for selecting an alternate movie. Only one allowed.
'rmvc' (page 2425)
Version criteria for selecting an alternate movie. Multiples allowed.
'rmcd' (page 2422)
Component availability information for selecting an alternate movie. Multiples allowed.
'rmqu' (page 2424)
Playback quality information for selecting an alternate movie. Only one allowed.
'rmla' (page 2423)
Language information for selecting an alternate movie. Only one allowed.
'rmcs' (page 2422)
CPU speed information for selecting an alternate movie. Only one allowed.

Discussion
The `'rdrf'` atom contains a `ReferenceMovieDataRefRecord`, which designates an alternate movie. The `'rmda'` atom's optional atoms help QuickTime decide whether or not to run that movie. If multiple `'rmvc'` or `'rmcd'` atoms are present, all their criteria must be satisfied for the movie to play.

See Also

See the `ReferenceMovieDataRefRecord` structure. For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'rmdr' Provides data rate information for selecting an alternate movie.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields
`atomType`

Discussion
Constant `ReferenceMovieDataRateAID`, designating atom type `'rmdr'`.

`data`

Discussion
A `QTAltDataRateRecord` data structure.

Parent Atom

'rmda' (page 2422)
Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'rmla' Provides language information for selecting an alternate movie.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields

`atomType`

Discussion

Constant `ReferenceMovieLanguageAID`, designating atom type `'rmla'`.

`data`

Discussion

A `QTAltLanguageRecord` structure.

Parent Atom

`'rmda'` (page 2422)

Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'rmqu' Provides playback quality information for selecting an alternate movie.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields

`atomType`

Discussion

Constant `ReferenceMovieQualityAID`, designating atom type `'rmqu'`.

`data`

Discussion

A quality value of type SInt32. Higher quality values are selected over lower quality values.

Parent Atom

`'rmda'` (page 2422)

Parent atom can contain only one atom of this type.

Discussion

If the criteria established by the `'rmdr'` (page 2423), `'rmvc'` (page 2425), and `'rmcd'` (page 2422) atoms are equally satisfied by two or more alternate movies, the one with the highest quality value will be selected.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'rmra' Designates a reference movie.

This is a QT atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameter:

Fields

`atomType`

Discussion

Constant `ReferenceMovieRecordAID`, designating atom type `'rmra'`.

Parent Atom

`'moov'` (page 2414)

Parent atom can contain only one atom of this type.

Required Child Atoms

`'rmda'` (page 2422)
Provides criteria for selecting an alternate movie. Only one allowed.

Discussion
You insert an `'rmra'` atom in a `'moov'` atom to create a reference movie. Each `'rmda'` atom in the `'rmra'` atom designates an alternate movie.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'rmvc' Provides version criteria for selecting an alternate movie.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields

`atomType`

Discussion
Constant `ReferenceMovieVersionCheckAID`, designating atom type `'rmvc'`.

`data`

Discussion
A `QTAltVersionCheckRecord` data structure.

Parent Atom

`'rmda'` (page 2422)
Parent atom can contain any number of atoms of this type.

Discussion
This optional atom in a `'rmda'` atom lets you demand minimum product version criteria for selecting an alternate movie. For example, a movie that needs QuickTime VR 2.1 or later could require a Gestalt `'qtvv'` value of 0x02100000 or higher.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'scpt' Transcript track reference type atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields

`atomType`

Discussion
Value is `'scpt'`.

`data`

Discussion
A list of track ID values (32-bit integers) specifying the related tracks. Note that a track ID value can be set to 0 to indicate an unused entry in the atom. Doing this can be more convenient than deleting the reference.

Parent Atom

`'tref'` (page 2442)
Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'sdp ' SDP text for a hint track.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields

`atomType`

Discussion
Value is `'sdp '`. The fourth character is a space.

`data`

Discussion
SDP text.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'sean' The outermost atom container, of which all other atoms are children.

This is a QT container atom; it is not declared in the header files. You can create it with `QTNewAtomContainer` (page 1451), using the following parameter:

Fields

`atomData`

Discussion
A pointer to a memory location that will hold the new atom.

Discussion
After creating a `'sean'` atom, you can populate it with other atoms by using `QTInsertChild` (page 1446).

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'SelO' User data list entry atom: play selection only.

```
struct MoviesUserData {
    long    size;
    long    udType;
    char    data[1];
};
```

Fields

`size`

Discussion
The size in bytes of this atom structure.

`udType`

Discussion
Constant `'SelO'`.

`data`

Discussion
A byte indicating that only the selected area of the movie should be played.

Parent Atom

`'udta'` (page 2446)

Parent atom can contain only one atom of this type.

Programming Info

C interface file: `MoviesFormat.h`

See Also

See the `MoviesUserData` function. For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'skip' Unused space available in the file.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields

`atomType`

Discussion

Constant `SkipAtomType`, designating atom type `'skip'`.

`data`

Discussion

Any number of bytes of free space.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'smhd' Contains sound stereo balance information.

```
struct SoundMediaInfoHeaderAtom {
    long                    size;
    long                    atomType;
    SoundMediaInfoHeader    smiHeader;
};
```

Fields

`size`

Discussion

The size in bytes of this atom structure.

`atomType`

Discussion

Constant `SoundMediaInfoHeaderAID`, designating atom type `'smhd'`.

`smiHeader`

Discussion

A `SoundMediaInfoHeader` data structure, which contains the actual data for this atom.

Discussion

The `SoundMediaInfoHeader` data structure currently contains only stereo balance information.

Programming Info

C interface file: `MoviesFormat.h`

See Also

For the structure that contains this atom, see `'minf'[sound]` (page 2412). For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'sprt' A key frame sprite definition.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields

`atomType`

Discussion

Constant `kSpriteAtomType`, designating atom type `'sprt'`.

`data`

Discussion

A list of sprite property constants (see below).

Sprite property constants

`kSpritePropertyMatrix`
(Value is 1). Describes the sprite's location and scaling within its sprite world or sprite track. By modifying a sprite's matrix, you can modify the sprite's location so that it appears to move in a smooth path on the screen or so that it jumps from one place to another. You can modify a sprite's size, so that it shrinks, grows, or stretches. Depending on which image compressor is used to create the sprite images, other transformations, such as rotation, may be supported as well. Translation-only matrixes provide the best performance.

`kSpritePropertyVisible`
(Value is 4). Specifies whether or not the sprite is visible. To make a sprite visible, you set the sprite's visible property to true.

`kSpritePropertyLayer`
(Value is 5). Contains a 16-bit integer value specifying the layer into which the sprite is to be drawn. Sprites with lower layer numbers appear in front of sprites with higher layer numbers. To designate a sprite as a background sprite, you should assign it the special layer number `kBackgroundSpriteLayerNum`.

`kSpritePropertyGraphicsMode`
(Value is 6). Specifies a graphics mode and blend color that indicates how to blend a sprite with any sprites behind it and with the background. To set a sprite's graphics mode, you call `SetSpriteProperty` (page 1491), passing a pointer to a `ModifierTrackGraphicsModeRecord` structure.

`kSpritePropertyActionHandlingSpriteID`
(Value is 8). Specifies another sprite by ID that delegates QT events.

`kSpritePropertyImageIndex`
(Value is 100). Contains the atom ID of the sprite's image atom.

`kSpriteUsesImageIDsAtomType`
(Value is `'uses'`). Lets a sprite specify the subset of images that `kSpritePropertyImageIndex` can refer to.

Parent Atom

`'stss'` (page 2435)

Parent atom can contain only one atom of this type.

Discussion

`Sprite` atoms should have ID numbers start at 1 and count consecutively upward.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'sptl' Specifies which graphics export compressor to use, its depth, and the spatial quality.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields
`size`

Discussion
The size in bytes of this atom structure.

`atomType`

Discussion
Constant `scSpatialSettingsType`, designating atom type `'sptl'`.

`data`

Discussion
A pointer to `SCSpatialSettings` structure.

See Also

See the `GraphicsExportCanUseCompressor` (page 975) function. For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'ssrc' Nonprimary source track reference type atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields
`atomType`

Discussion
Constant `kTrackModifierReference`, designating atom type `'ssrc'`.

`data`

Discussion
A list of track ID values (32-bit integers) specifying the related tracks. Note that a track ID value can be set to 0 to indicate an unused entry in the atom. Doing this can be more convenient than deleting the reference.

Parent Atom

`'tref'` (page 2442)
Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'sstr' Specifies the ID of a string variable, contained in a sprite track, to display in the status area of the browser.

This is a QT container atom; it is not declared in the header files. You can create it with `QTNewAtomContainer` (page 1451) and insert it with `QTInsertChild` (page 1446), using the following parameter:

Fields
`atomType`

Discussion
Value is `'sstr'`.

Parent Atom

'beha' (page 2373)

Defines sprite behavior.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'stbl' Contains information for converting from media time to sample number to sample location and indicates how to interpret samples and chunks.

```
struct SampleTableAtom {
    long                        size;
    long                        atomType;
    SampleDescriptionAtom       sampleDescription;
    TimeToSampleNumAtom         timeToSampleNum;
    SampleToChunkAtom           sampleToChunk;
    SyncSampleAtom              syncSample;
    SampleSizeAtom              sampleSize;
    ChunkOffsetAtom             chunkOffset;
    ShadowSyncAtom              shadowSync;
};
```

Fields

`size`

Discussion

The size in bytes of this atom structure.

`atomType`

Discussion

Constant `SampleTableAID`, designating atom type `'stbl'`.

`sampleDescription`

Discussion

A 'stsd' (page 2433) atom, which contains information required to decode the samples in the media.

`timeToSampleNum`

Discussion

A 'stts' (page 2437) atom that relates sample numbers to sample durations.

`sampleToChunk`

Discussion

A 'stsc' (page 2433) atom that maps sample numbers to chunk numbers.

`syncSample`

Discussion

A 'stss' (page 2435) atom that identifies the key frames in the media.

`sampleSize`

Discussion

A 'stsz' (page 2436) atom, which identifies the size of each sample in the media.

`chunkOffset`

Discussion

A 'stco' (page 2431) atom that identifies the location of each data chunk in the media.

shadowSync

Discussion
A 'stsh' (page 2434) atom, which lists self-contained sync samples that are alternates for existing frame difference samples. This field may be omitted.

Discussion
This atom contains the information you need to find a sample number based on a time and to find the sample's location based on the sample number. Samples are organized into chunks, containing one or more samples. This atom contains the information you need to find out which chunk holds a given sample, where that chunk begins, and where in that chunk you can find the sample.

Programming Info
C interface file: MoviesFormat.h

See Also

For the atoms that may contain this atom, see 'minf'[generic] (page 2411), 'minf'[sound] (page 2412), and 'minf'[video] (page 2413). For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'stco' Identifies the location of each chunk of data in the media's data stream.

```
struct ChunkOffsetAtom {
    long    size;
    long    atomType;
    long    flags;
    long    numEntries;
    long    chunkOffsetTable[1];
};
```

Fields
size

Discussion
The size in bytes of this atom structure.

atomType

Discussion
Constant STChunkOffsetAID, designating atom type 'stco'.

flags

Discussion
One byte of version information followed by three bytes of flags. The flags bytes are not currently used.

numEntries

Discussion
The number of entries in chunkOffsetTable.

chunkOffsetTable

Discussion
An array of chunk offset values.

Discussion
A chunk is a collection of data samples in a media that allows optimized data access. A chunk may contain one or more samples. Chunks in a media may have different sizes, and the samples within a chunk may have different sizes.

Programming Info
C interface file: `MoviesFormat.h`

See Also

For the structure that contains this atom, see `'stbl'` (page 2430). For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'strt' Defines the starting offset of hypertext in a text stream.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields
`atomType`

Discussion
Value is `'strt'`.

`data`

Discussion
The starting offset of hypertext.

Parent Atom

`'htxt'` (page 2399)
Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'strv' Contains a string variable for a sprite.

This is a QT container atom; it is not declared in the header files. You can create it with `QTNewAtomContainer` (page 1451) and insert it with `QTInsertChild` (page 1446), using the following parameter:

Fields
`atomType`

Discussion
Value is `'strv'`.

Parent Atom

`'vars'` (page 2448)
Contains variables for a sprite.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'stsc' Maps sample numbers to chunk numbers.

```
struct SampleToChunkAtom {
    long            size;
    long            atomType;
    long            flags;
    long            numEntries;
    SampleToChunk   sampleToChunkTable[1];
};
```

Fields

`size`

Discussion

The size in bytes of this atom structure.

`atomType`

Discussion

Constant `STSampleToChunkAID`, designating atom type `'stsc'`.

`flags`

Discussion

One byte of version information followed by three bytes of flags. The flags bytes are not currently used.

`numEntries`

Discussion

The number of entries in `sampleToChunkTable`.

`sampleToChunkTable`

Discussion

An array of `SampleToChunk` data structures, which contain the actual data for this atom.

Discussion

A chunk is a collection of data samples in a media that allows optimized data access. A chunk may contain one or more samples. Chunks in a media may have different sizes, and the samples within a chunk may have different sizes.

Programming Info

C interface file: `MoviesFormat.h`

See Also

For the structure that contains this atom, see `'stbl'` (page 2430). For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'stsd' Holds one or more sample description structures.

```
struct SampleDescriptionAtom {
    long                size;
    long                atomType;
    long                flags;
    long                numEntries;
    SampleDescription   sampleDescTable[1];
};
```

Fields

`size`

Discussion

The size in bytes of this atom structure.

atomType

Discussion
Constant STSampleDescAID, designating atom type 'stsd'.

flags

Discussion
One byte of version information followed by three bytes of flags. The flags bytes are not currently used.

numEntries

Discussion
Number of entries in sampleDescTable.

sampleDescTable

Discussion
An array of SampleDescription data structures, which contain the actual data for this atom.

Discussion
In QuickTime streaming, this atom describes the data format of the hint samples and contains track-level information, such as the RTP timescale for a track.

Programming Info
C interface file: MoviesFormat.h

See Also

For the structure that contains this atom, see 'stbl' (page 2430). For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'stsh' Lists self-contained sync samples that are alternates for existing frame difference samples.

```
struct ShadowSyncAtom {
    long           size;
    long           atomType;
    long           flags;
    long           numEntries;
    ShadowSync     shadowSyncTable[1];
};
```

Fields
size

Discussion
The size in bytes of this atom structure.

atomType

Discussion
Constant STShadowSyncAID, designating atom type 'stsh'.

flags

Discussion
One byte of version information followed by three bytes of flags. The flags bytes are not currently used.

numEntries

Discussion
The number of entries in shadowSyncTable.

`shadowSyncTable`

Discussion
An array of `ShadowSync` data structures, which contain the actual data for this atom.

Discussion
Shadow sync atoms are used to optimize random access operations on a movie, thereby enhancing playback performance.

Programming Info
C interface file: `MoviesFormat.h`

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'stss' Identifies the key frames in a media.

```
struct SyncSampleAtom {
    long    size;
    long    atomType;
    long    flags;
    long    numEntries;
    long    syncSampleTable[1];
};
```

Fields
`size`

Discussion
The size in bytes of this atom structure.

`atomType`

Discussion
Constant `STSyncSampleAID`, designating atom type `'stss'`.

`flags`

Discussion
Flags (currently not used).

`numEntries`

Discussion
The number of entries in `syncSampleTable`.

`syncSampleTable`

Discussion
An array of physical sample numbers, each of which identifies a key frame in the media.

Discussion
In a media that contains compressed data, key frames define starting points for portions of a temporally compressed sequence. Each key frame is independent of the preceding frames. Subsequent frames may depend on the key frame.

Programming Info
C interface file: `MoviesFormat.h`

See Also

For the structure that contains this atom, see `'stbl'` (page 2430). For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'stsz' Identifies the size of each sample in a media.

```
struct SampleSizeAtom {
    long    size;
    long    atomType;
    long    flags;
    long    sampleSize;
    long    numEntries;
    long    sampleSizeTable[1];
};
```

Fields

`size`

Discussion

The size in bytes of this atom structure.

`atomType`

Discussion

Constant `STSampleSizeAID`, designating atom type `'stsz'`.

`flags`

Discussion

One byte of version information followed by three bytes of flags. The flags bytes are not currently used.

`sampleSize`

Discussion

The number of bytes in the samples. If all the samples are the same size, `sampleSize` indicates the size of all the samples. If `sampleSize` is set to 0, then the samples have different sizes, and those sizes are stored in `sampleSizeTable`.

`numEntries`

Discussion

The number of entries in `sampleSizeTable`.

`sampleSizeTable`

Discussion

An array of numbers, one for every sample. This field is indexed by sample number; the first entry corresponds to the first sample, the second to the second sample, and so on.

Programming Info

C interface file: `MoviesFormat.h`

See Also

For the structure that contains this atom, see `'stbl'` (page 2430). For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'stts' Holds one or more structures that relate sample numbers to sample durations.

```
struct TimeToSampleNumAtom {
    long                size;
    long                atomType;
    long                flags;
    long                numEntries;
    TimeToSampleNum     timeToSampleNumTable[1];
};
```

Fields
`size`

Discussion
The size in bytes of this atom structure.

`atomType`

Discussion
Constant `STTimeToSampAID`, designating atom type `'stts'`.

`flags`

Discussion
One byte of version information followed by three bytes of flags. The flags bytes are not currently used.

`numEntries`

Discussion
The number of entries in `timeToSampleNumTable`.

`timeToSampleNumTable`

Discussion
An array of `TimeToSampleNum` data structures, each of which maps a sample number to its sample duration.

Discussion
Entries in `timeToSampleNumTable` collect samples according to their order in the media and their duration. If consecutive samples have the same duration, a single table entry may be used to define more than one sample. In these cases, the `count` field indicates the number of consecutive samples that have the same duration. For example, if a video media had a constant frame rate, `timeToSampleNumTable` would have one entry.

Programming Info
C interface file: `MoviesFormat.h`

See Also

For the structure that contains this atom, see `'stbl'` (page 2430). For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'sync' Synchronization track reference type atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields
`atomType`

Discussion
Value is `'sync'`.

`data`

Discussion
A list of track ID values (32-bit integers) specifying the related tracks. Note that a track ID value can be set to 0 to indicate an unused entry in the atom. Doing this can be more convenient than deleting the reference.

Parent Atom

`'tref'` (page 2442)
Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'tbox' Defines a text box rectangle.

```
struct TextBoxAtom {
    long    size;
    long    atomType;
    Rect    textBox;
};
```

Fields
`size`

Discussion
The size in bytes of this atom structure.

`atomType`

Discussion
Value is `'tbox'`.

`textBox`

Discussion
A new text box rectangle, which overrides the rectangle defined by the `defaultTextBox` constant.

Discussion
This is a classic atom; you can access its information by calculating offsets.

Programming Info
C interface file: `MoviesFormat.h`

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'tcmi' Time code media information atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields
`atomType`

Discussion
Value is `'tcmi'`.

`data`

Discussion
Time code media information.

Parent Atom

'minf'[video] (page 2413)

Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'tkhd' Specifies the characteristics of a track in a movie.

```
struct TrackHeaderAtom {
    long            size;
    long            atomType;
    TrackHeader     header;
};
```

Fields

`size`

Discussion

The size in bytes of this atom structure.

`atomType`

Discussion

Constant `TrackHeaderAID`, designating atom type `'tkhd'`.

`header`

Discussion

A `TrackHeader` structure that contains the actual data for this atom.

Parent Atom

'trak' (page 2441)

Parent atom can contain only one atom of this type.

Programming Info

C interface file: `MoviesFormat.h`

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'tmax' Largest relative transmission time, in milliseconds.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields

`atomType`

Discussion

Value is `'tmax'`.

`data`

Discussion

4 bytes.

Parent Atom

'hinf' (page 2396)

Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'tmcd' Time code track reference type atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields

`atomType`

Discussion

Constant `TimeCodeMediaType`, designating atom type `'tmcd'`.

`data`

Discussion

A list of track ID values (32-bit integers) specifying the related tracks. Note that a track ID value can be set to 0 to indicate an unused entry in the atom. Doing this can be more convenient than deleting the reference.

Parent Atom

`'tref'` (page 2442)

Parent atom can contain only one atom of this type.

See Also

See the `TimeCodeDescription` structure. For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'tmin' Smallest relative transmission time, in milliseconds.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields

`atomType`

Discussion

Value is `'tmin'`.

`data`

Discussion

4 bytes.

Parent Atom

`'hinf'` (page 2396)

Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'tpyl' Total number of bytes that will be sent, not including 12-byte RTP headers.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields

`atomType`

Discussion

Value is `'tpyl'`.

`data`

Discussion
8 bytes.

Parent Atom

`'hinf'` (page 2396)
Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'trak' Defines a single track of a movie.

```
struct TrackDirectory {
    long                size;
    long                atomType;
    TrackHeaderAtom     trackHeader;
    ClippingAtom        trackClip;
    EditsAtom           edits;
    MediaDirectory      media;
    UserDataAtom        userData;
};
```

Fields
`size`

Discussion
The size in bytes of this atom structure.

`atomType`

Discussion
Constant `TrackAID`, designating atom type `'trak'`.

`trackHeader`

Discussion
A `'tkhd'` (page 2439) atom, which specifies general characteristics of the track.

`trackClip`

Discussion
A `'clip'` (page 2373) atom, which defines the track's clipping region.

`edits`

Discussion
An `'edts'` (page 2389) atom, which defines the portions of the track's media that are going into the track.

`media`

Discussion
A `'mdia'` (page 2409) atom structure that defines the media for the track.

`userData`

Discussion
A `'udta'` (page 2446) atom that contains user data.

Parent Atom

`'moov'` (page 2414)
Parent atom can contain any number of atoms of this type.

Required Child Atoms

`'tkhd'` (page 2439)
Specifies general characteristics of the track. Only one allowed.
`'mdia'` (page 2409)
The media for the track. Only one allowed.

Optional Child Atoms

`'clip'` (page 2373)
Defines the clipping region for the track. Only one allowed.
`'matt'` (page 2407)
Defines a matte for a track's media. Only one allowed.
`'edts'` (page 2389)
Defines the portions of the track's media that are going into the track. Only one allowed.
`'tref'` (page 2442)
`Track` reference atom. Only one allowed.
`'load'` (page 2406)
Contains preloading information for a track. Only one allowed.
`'imap'` (page 2401)
An input map. Only one allowed.
`'udta'` (page 2446)
User data atom. Only one allowed.

Discussion
A movie may consist of one or more tracks. Each track is independent of the other tracks in the movie and carries its own temporal and spatial information. Each track atom contains an associated media atom. Note that there must be at least one media track within a QuickTime movie. All media tracks that are present must remain in the movie, even if the media data within them is not referenced by the hint tracks. After deleting all hint tracks, the entire unhinted movie must remain.

Programming Info
C interface file: `MoviesFormat.h`

See Also

See the `TrackDirectoryEntry` structure. For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'tref' `Track` reference atom.

This is a QT container atom; it is not declared in the header files. You can create it with `QTNewAtomContainer` (page 1451) and insert it with `QTInsertChild` (page 1446), using the following parameters:

Fields
`atomType`

Discussion
Constant `TrackHeaderAID`, designating atom type `'tref'`.

`data`

Discussion
One track reference atom of a type listed below.

Parent Atom

`'trak'` (page 2441)
Parent atom can contain only one atom of this type.

Required Child Atoms (one from this list)

`'tmcd'` (page 2440)
Time code track reference type atom. Only one allowed.
`'chap'` (page 2373)
Chapter or scene list track reference type atom. Only one allowed.
`'sync'` (page 2437)
Synchronization track reference type atom. Only one allowed.
`'scpt'` (page 2425)
Transcript track reference type atom. Only one allowed.
`'ssrc'` (page 2429)
Nonprimary source track reference type atom. Only one allowed.
`'hint'` (page 2396)
Hint track reference type atom. Only one allowed.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'trpy' Total number of bytes that will be sent, including 12-byte RTP headers, but not including network headers.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields
`atomType`

Discussion
Value is `'trpy'`.

`data`

Discussion
8 bytes.

Parent Atom

`'hinf'` (page 2396)
Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'twdt' Tween data type atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields
`atomType`

Discussion
Value is `'twdt'`.

`data`

Discussion
Tween data.

Parent Atom

`'twen'` (page 2444)

Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'twdu' Tween duration atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields

`atomType`

Discussion

Constant `kTweenDuration`, designating atom type `'twdu'`.

`data`

Discussion

Tween duration data.

Parent Atom

`'twen'` (page 2444)

Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'twen' Tween entry atom.

This is a QT container atom; it is not declared in the header files. You can create it with `QTNewAtomContainer` (page 1451) and insert it with `QTInsertChild` (page 1446), using the following parameter:

Fields

`atomType`

Discussion

Constant `kTweenEntry`, designating atom type `'twen'`.

Required Child Atoms

`'twst'` (page 2445)

Tween start atom. Only one allowed.

`'twdu'` (page 2444)

Tween duration atom. Only one allowed.

`'twdt'` (page 2443)

Tween data type atom. Only one allowed.

`'twnt'` (page 2444)

Tween type atom. Only one allowed.

See Also

See the `TweenSequenceEntryRecord` structure. For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'twnt' Tween type atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields
`atomType`

Discussion
Constant `kTweenType`, designating atom type `'twnt'`.

`data`

Discussion
A tween type; see `Tween Types`.

Parent Atom

`'twen'` (page 2444)
Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'twst' Tween start offset atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields
`atomType`

Discussion
Constant `kTweenStartOffset`, designating atom type `'twst'`.

`data`

Discussion
Tween start offset.

Parent Atom

`'twen'` (page 2444)
Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

' ty' Input atom type.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields
`atomType`

Discussion
Value is `'  ty'`; first and second characters are spaces.

`data`

Discussion
`Track` input atom type code.

Parent Atom

`'  in'` (page 2405)
Parent atom can contain only one atom of this type.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'udta' Holds one or more structures of movie user data.

```
struct UserDataAtom {
    long              size;
    long              atomType;
    MoviesUserData    userData[1];
};
```

Fields

`size`

Discussion

The size in bytes of this atom structure.

`atomType`

Discussion

Constant `UserDataAID`, designating atom type `'udta'`.

`userData`

Discussion

An array of `MoviesUserData` data structures, which contain the actual data for this atom. The currently defined types are listed below.

Parent atom (one or the other)

`'moov'` (page 2414)
Parent atom can contain any number of atoms of this type.
`'trak'` (page 2441)
Parent atom can contain any number of atoms of this type.

Optional child atom

`' ccpy'` (page 2375)
Copyright statement.
`' cday'` (page 2376)
Date the movie content was created.
`' cdir'` (page 2377)
Name of movie's director.
`' ced1'` (page 2377)
First edit date and description. Similar for ' ced2' through ' ced9'.
`' cfmt'` (page 2378)
Indication of movie format (computer-generated, digitized, and so on).
`' cinf'` (page 2378)
Information about the movie.
`' cprd'` (page 2379)
Name of movie's producer.
`' cprf'` (page 2379)
Names of performers.
`' creq'` (page 2380)
Special hardware and software requirements.
`' csrc'` (page 2381)
Credits for those who provided movie source content.

`' cwrt'` (page 2381)
Name of movie's writer.
`'WLOC'` (page 2453)
Default window location for movie. Two 16-bit values, `{x,y}`.
`'name'[userdata]` (page 2416)
Name of object.
`'LOOP'` (page 2407)
Looping. Long integer indicating looping `style`. 0 for none, 1 for looping, 2 for palindromic looping.
`'SelO'` (page 2426)
Play selection only. `Byte` indicating that only the selected area of the movie should be played.
`'AllF'` (page 2372)
Play all frames. `Byte` indicating that all frames of video should be played, regardless of timing.

Discussion
Inside the user data atom is a list of atoms describing each piece of user data. Each data element contains size and type information along with the data. Furthermore, for historical reasons, the list of atoms is optionally terminated by a 32-bit integer set to 0. If you are writing a program to read user data atoms, you should allow for the terminating 0. However, if you are writing a program to create user data atoms, you can safely leave out the trailing 0.

Programming Info
C interface file: `MoviesFormat.h`

See Also

See the `MoviesUserData` structure. For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'url ' Contains a URL for a sprite.

This is a QT container atom; it is not declared in the header files. You can create it with `QTNewAtomContainer` (page 1451) and insert it with `QTInsertChild` (page 1446), using the following parameter:

Fields
`atomType`

Discussion
Value is `'url '`.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'uses' Lets a sprite specify the subset of images that its image index property can refer to.

This is a QT container atom; it is not declared in the header files. You can create it with `QTNewAtomContainer` (page 1451) and insert it with `QTInsertChild` (page 1446), using the following parameter:

Fields
`atomType`

Discussion
Value is `'uses'`.

Parent Atom

`'sprt'` (page 2428)
A key frame sprite definition.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'vars' Contains variables for a sprite.

This is a QT container atom; it is not declared in the header files. You can create it with `QTNewAtomContainer` (page 1451) and insert it with `QTInsertChild` (page 1446), using the following parameter:

Fields
`atomType`

Discussion
Value is `'vars'`.

Optional Child Atoms

`'flov'` (page 2393)
Contains a floating-point variable for a sprite. Multiple atoms allowed.
`'strv'` (page 2432)
Contains a string variable for a sprite. Multiple atoms allowed.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'vide' Contains compression information for the Base Image Exporter.

This is a QT atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameter:

Fields
`atomType`

Discussion
Constant `VideoMediaType`, designating atom type `'vide'`; see `Component Identifiers`.

Optional Child Atoms

`'dasz'` (page 2385)
Only one allowed. If present, it specifies a desired data size. The base exporter repeats compression attempts, decreasing the `quality` parameter until it reaches this size or lower, or it runs out of patience.
`'reso'` (page 2421)
Only one allowed. If present, it specifies the resolution for the `pixmap` passed to the compressor.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'vmhd'[media] Stores handler-specific information for video media in a track.

```
struct VideoMediaInfo {
    long                            size;
    long                            atomType;
    VideoMediaInfoHeaderAtom        header;
    HandlerAtom                     dataHandler;
    DataInfoAtom                    dataInfo;
    SampleTableAtom                 sampleTable;
};
```

Fields
`size`

Discussion
The size in bytes of this atom structure.

atomType

Discussion
Constant VideoMediaInfoAID, designating atom type 'vmhd'.

header

Discussion
A 'vmhd'[transfer] (page 2449) atom, which defines the graphics transfer mode for this video media.

dataHandler

Discussion
A 'hdlr' (page 2395) atom that specifies the component that is to handle this media.

dataInfo

Discussion
A 'dinf' (page 2387) atom, which specifies where the video media data is stored.

sampleTable

Discussion
A 'stbl' (page 2430) atom, which tells the media handler how to locate and interpret data samples.

Discussion
This atom stores handler-specific information for the media that constitutes a video track. A video media handler uses this information to map from media time to media data. Another type of media handler would not know how to interpret this information.

Programming Info
C interface file: MoviesFormat.h

See Also

See the 'minf'[sound] (page 2412) atom for sound media and the 'minf'[generic] (page 2411) atom for media other than video or sound. Also see the VideoMediaInfoHeader structure. For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'vmhd'[transfer] Defines the graphics transfer characteristics for a video media.

```
struct VideoMediaInfoHeaderAtom {
    long                    size;
    long                    atomType;
    VideoMediaInfoHeader    vmiHeader;
};
```

Fields
size

Discussion
The size in bytes of this atom structure.

atomType

Discussion
Constant VideoMediaInfoHeaderAID, designating atom type 'vmhd'.

vmiHeader

Discussion
A VideoMediaInfoHeader data structure, which contains the actual data for this atom.

Programming Info
C interface file: `MoviesFormat.h`

See Also

For the structure that contains this atom, see `'minf'[video]` (page 2413). Also see the `VideoMediaInfoHeader` structure. For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'vrcp' Custom cursor atom parent.

This is a QT container atom; it is not declared in the header files. You can create it with `QTNewAtomContainer` (page 1451) and insert it with `QTInsertChild` (page 1446), using the following parameter:

Fields
`atomType`

Discussion
Constant `kQTVRCursorParentAtomType`, designating atom type `'vrcp'`.

Required Child Atoms

`'CURS'` (page 2384)
Custom cursor child atom. Multiple atoms of this type allowed.
`'crsr'` (page 2382)
Color custom cursor child atom. Multiple atoms of this type allowed.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'vrni' QTVR node ID atom.

This is a QT container atom; it is not declared in the header files. You can create it with `QTNewAtomContainer` (page 1451) and insert it with `QTInsertChild` (page 1446), using the following parameter:

Fields
`atomType`

Discussion
Constant `kQTVRNodeIDAtomType`, designating atom type `'vrni'`.

Parent Atom

`'vrnp'` (page 2450)
Parent atom can contain any number of atoms of this type.

Required Child Atoms

`'nloc'` (page 2417)
QTVR node location atom. Only one allowed.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'vrnp' QTVR node parent atom.

This is a QT container atom; it is not declared in the header files. You can create it with `QTNewAtomContainer` (page 1451) and insert it with `QTInsertChild` (page 1446), using the following parameter:

Fields
`atomType`

Discussion
Constant `kQTVRNodeParentAtomType`, designating atom type `'vrnp'`.

Required Child Atoms

`'vrni'` (page 2450)

QTVR node ID atom. Any number allowed.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'vrsc' VR world header atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields

`atomType`

Discussion

Constant `kQTVRWorldHeaderAtomType`, designating atom type `'vrsc'`.

`data`

Discussion

Contains the name of the scene and the default node ID.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'vrsg' Contains the name of a VR scene.

This is a QT leaf atom that contains a struct in its data field. Using the constant `kQTVRStringAtomType` and a pointer to the atom, you can access its data with `QTGetAtomDataPtr` (page 1433) and change it with `QTSetAtomData` (page 1465). The struct is declared as follows:

```
struct QTVRStringAtom {
    UInt16          stringUsage;
    UInt16          stringLength;
    unsigned char   theString[4];
};
```

Fields

`stringUsage`

Discussion

Unused field.

`stringLength`

Discussion

The length of the name string in bytes.

`theString`

Discussion

The name as a C string.

Discussion

One leaf atom of this type is contained in a VR world container. You can get a pointer to this container by calling `QTVRGetVRWorld` (page 2019). One of this atom's siblings in the VR world is a `'vrsc'` (page 2451) atom, which contains the atom ID of this atom in its `nameAtomID` field. The following code illustrates a function that returns the name of a VR node as a Pascal string, given the node's ID:

```
OSErr MyGetNodeName (QTVRInstance theInstance, UInt32 theNodeID,
                     StringPtr theStringPtr)
{
    OSErr                       theErr =noErr;
    QTAtomContainer             theNodeInfo;
    QTVRNodeHeaderAtomPtr       theNodeHeader;
    QTAtom                      theNodeHeaderAtom =0;

    // Get the node information atom container
    theErr =QTVRGetNodeInfo(theInstance, theNodeID, &theNodeInfo);

    // Get the node header atom.
    if (!theErr)
        theNodeHeaderAtom =QTFindChildByID(theNodeInfo,
                                           kParentAtomIsContainer,
                                            kQTVRNodeHeaderAtomType, 1, nil);
    if (theNodeHeaderAtom !=0) {
        QTLockContainer(theNodeInfo);

        // Get a pointer to the node header atom data.
        theErr =QTGetAtomDataPtr(theNodeInfo, theNodeHeaderAtom, nil,
                                  (Ptr *)&theNodeHeader);
        // See if there is a name atom.
        if (!theErr && theNodeHeader->
nameAtomID !=0) {
            QTAtom theNameAtom;
            theNameAtom =QTFindChildByID(theNodeInfo,
                          kParentAtomIsContainer, kQTVRStringAtomType,
                          theNodeHeader->
nameAtomID, nil);
            if (theNameAtom !=0) {
                VRStringAtomPtr theStringAtomPtr;

                // Get a pointer to the name atom data; copy it into string
                theErr =QTGetAtomDataPtr(theNodeInfo, theNameAtom, nil,
                                          (Ptr *)&theStringAtomPtr);
                if (!theErr) {
                    short theLen =theStringAtomPtr->
stringLength;
                    if (theLen >
255)
                        theLen =255;
                    BlockMove(theStringAtomPtr->
theString,
                              &theStringPtr[1], theLen);
                    theStringPtr[0] =theLen;
                }
            }
        }
        QTUnlockContainer(theNodeInfo);
    }
    QTDisposeAtomContainer(theNodeInfo);
    return(theErr);
}
```

Programming Info
C interface file: `QuickTimeVRFormat.h`

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'wide' Wide atom name placeholder atom.

This is a QT leaf atom; it is not declared in the header files. You can create it with `QTInsertChild` (page 1446) using the following parameters:

Fields
`atomType`

Discussion
Constant `WideAtomPlaceholderType`, designating atom type `'wide'`.

`data`

Discussion
8 bytes of placeholder space to allow an atom to be converted from a 32-bit to a 64-bit atom.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'WLOC' User data list entry atom: default window location for movie.

```
struct MoviesUserData {
    long    size;
    long    udType;
    char    data[1];
};
```

Fields
`size`

Discussion
The size in bytes of this atom structure.

`udType`

Discussion
Value is `'WLOC'`.

`data`

Discussion
2 16-bit values, `{x,y}`.

Parent Atom

`'udta'` (page 2446)
Parent atom can contain only one atom of this type.

Programming Info
C interface file: `MoviesFormat.h`

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

'wtxt' Parent atom for hypertext items.

This is a QT container atom; it is not declared in the header files. You can create it with `QTNewAtomContainer` (page 1451) and insert it with `QTInsertChild` (page 1446), using the following parameter:

Fields

`atomType`

Discussion

Value is `'wtxt'`.

Required Child Atoms

`'strt'` (page 2432)

Defines the starting offset of hypertext in a text stream. Only one allowed.

`'end '` (page 2391)

Defines the ending offset of hypertext in a text stream. Only one allowed.

Optional Child Atoms

`'htxt'` (page 2399)

Multiple atoms allowed.

See Also

For general information about atoms, see *Inside QuickTime: QuickTime File Format*.

QuickTime Public Resources

Resources

'atms'

Lists effect and parameter description atoms for effect components.

```
type 'atms' {
    longint;                                        // root atom count
    array AtomArray {
        literal    longint;                         // atomType
        longint;                                    // atomID
        longint    noChildren =0;                  // children
        longint =$$CountOf(AtomData);
        array AtomData {
            switch {
                case long:
                    key literal longint ='long';
                    pstring;                        // data
                case short:
                    key literal longint ='shrt';
                    pstring;                        // data
                case noMininumFixed:
                    key literal longint ='nmiF';
                    pstring = "";                   // data
                case noMaximumFixed:
                    key literal longint ='nmaF';
                    pstring = "";                   // data
                case noMininumDouble:
                    key literal longint ='nmiD';
                    pstring = "";                   // data
                case noMaximumDouble:
                    key literal longint ='nmaD';
                    pstring = "";                   // data
                case fixed:
                    key literal longint ='fixd';
                    pstring;                        // data
                case double:
                    key literal longint ='doub';
                    pstring;                        // data
                case string:
                    key literal longint ='str ';
                    pstring;                        // data
                case lstring:
                    key literal longint ='lstr';
                LongStringStart:
                    longint =
                    ((LongStringEnd[$$ArrayIndex(AtomArray),
                        $$ArrayIndex(AtomData)] -
                        LongStringStart[$$ArrayIndex(AtomArray),
                        $$ArrayIndex(AtomData)]) >
>
3) - 4;
                    hex string
                        [$$Word(LongStringStart[$$ArrayIndex(AtomArray),
                        $$ArrayIndex(AtomData)]) - 4];
                LongStringEnd:
                case OSType:
                    key literal longint ='osty';
                    pstring;                        // data
            };
        };
    };
};
```

Discussion
The 'atms' resource for a video effect contains two sets of information. The first set contains the effect information that is used to construct the standard parameters dialog box. This includes items such as the name of the effect and optional copyright information. The second set contains a description of each parameter that the effect takes. If the effect does not take parameters, there is no information in this set.

Version Notes
Introduced in QuickTime 6.

Programming Info
Resource accessibility: Can be made public
Rez source file: `ImageCodec.r`
Programming summary: `Component Public Resources`

'avvc'

Lists AVI four cc types for compressor components.

```
type 'avvc' {
    array {
        literal    longint;    // avi four cc type
    };
};
```

Version Notes
Introduced in QuickTime 6.

Programming Info
Resource accessibility: Can be made public
Rez source file: `ImageCodec.r`
Programming summary: `Component Public Resources`

'avvd'

Lists AVI four cc types for decompressor components.

```
type 'avvd' {
    array {
        literal    longint;    // avi four cc type
    };
};
```

Version Notes
Introduced in QuickTime 6.

Programming Info
Resource accessibility: Can be made public
Rez source file: `ImageCodec.r`
Programming summary: `Component Public Resources`

'cdci'

Contains codec characteristics.

```
type 'cdci' {
    pstring[31];
    hex integer    version;
    hex integer    revlevel;
    hex longint    vendor;
    hex longint    decompressFlags;
    hex longint    compressFlags;
    hex longint    formatFlags;
    byte           compressionAccuracy;
    byte           decompressionAccuracy;
    integer        compressionSpeed;
    integer        decompressionSpeed;
    byte           compressionLevel;
    byte           resvd;
    integer        minimumHeight;
    integer        minimumWidth;
    integer        decompressPipelineLatency;
    integer        compressPipelineLatency;
    longint        privateData;
};
```

Version Notes
Introduced in QuickTime 6.

Programming Info
Resource accessibility: Private
Rez source file: `ImageCodec.r`
Programming summary: `Component Public Resources`

'cdec'

Contains a codec string.

```
type 'cdec' {
    hex    string;
};
```

Version Notes
Introduced in QuickTime 6.

Programming Info
Resource accessibility: Private
Rez source file: `ImageCodec.r`
Programming summary: `Component Public Resources`

'cpix'

Lists supported pixel formats for a codec compressor.

```
type 'cpix' {
    array {
        literal    longint;
    };
};
```

Discussion
A `'cpix'` resource is an array of 4-character codes (such as '2vuy' or `'yuvs'`) that define the pixel formats a codec can accept. You can use this array to list any pixel formats that your codec prefers to straight RGB. An application can get the list, by calling `GetComponentPublicResource`, to see what kind of graphics world it should construct.

Version Notes
Introduced in QuickTime 6.

Programming Info
Resource accessibility: Can be made public
Rez source file: `ImageCodec.r`
Programming summary: `Component Public Resources`

'dlle'

Contains a string for a multiplatform component.

```
type 'dlle' {
    cstring;
};
```

Version Notes
Introduced in QuickTime 6.

Programming Info
Resource accessibility: Private
Rez source file: `Components.r`
Programming summary: `Component Public Resources`

'mcfg'

Lists characteristics of files supported by a graphics importer component.

```
type 'mcfg' {
    longint =kQTMediaConfigResourceVersion;   // resource version (long)
    // version of the component this applies to
    longint kVersionDoesntMatter =0;
    // array, one entry for each media type
    longint =$$Countof(MIMEInfoArray);
    array MIMEInfoArray {
        literal longint;  // ID of the group this type belongs with:
                          // OSType, a kQTMediaConfigStreamGroupID, etc.
        literal longint;  // MIME config flags:
                          // unsigned long, a kQTMediaConfigCanUseApp, etc.
        literal longint;  // MacOS file type when saved (OSType)
        literal longint;  // MacOS file creator when saved (OSType)
        literal longint;         // component type (OSType)
        literal longint;         // component subtype (OSType)
        literal longint;         // component manufacturer (OSType)
        unsigned hex longint;    // component flags
        // component flags mask
        unsigned hex longint kAnyComponentFlagsMask =0;
        literal longint;         // default file extension (OSType)
                                 // all caps to match subType
                                 // of eat and grip components
        literal longint;         // QT file group:
                                 // OSType, a kQTMediaInfoNetGroup, etc.
        longint =$$Countof(QTMediaSynonymsArray);
        array QTMediaSynonymsArray {
            pstring;             // array of media type synonyms
        };
        align long;              // align
        wide array [5] {
            pstring;
            // array of 5 Pascal strings:
            //     + media type description
            //     + file extension(s)
            //     + opening application name
            //     + missing software description
            //     + vendor info string (copyright, version, etc)
        };
        align long;              // align
        // array of MIME types that describe this
        // (eg. audio/mpeg, audio/x-mpeg, etc.)
        longint =$$Countof(MIMETypeArray);
        array MIMETypeArray {
            pstring;
        };
        align long;              // align
    };
};
```

Version Notes
Introduced in QuickTime 6.

Programming Info
Resource accessibility: Private
Rez source file: `QuicktimeComponents.r`
Programming summary: `Component Public Resources`

'mgrp'

Lists MIME groups supported by a graphics importer component.

```
type 'mgrp' {
    longint =kQTMediaGroupResourceVersion;    // resource version (long)
    // component version this applies to
    longint kVersionDoesntMatter =0;
    // array of group information
    // (optional unless you are defining new group(s))
    longint =$$Countof(MIMEGroupArray);
    array MIMEGroupArray {
        literal longint;                // group ID (OSType)
        pstring;                        // name of the grouping
        pstring;                        // description
        align long;                     // align
    };
};
```

Version Notes
Introduced in QuickTime 6.

Programming Info
Resource accessibility: Private
Rez source file: `QuicktimeComponents.r`
Programming summary: `Component Public Resources`

'mime'[resource]

Lists MIME types supported by a movie importer or exporter component.

```
type 'mime' {
    longint =0;    // 10 bytes of reserved
    longint =0;
    integer =0;
    integer =0;    // 2 bytes of lock count
    parentStart:
        longint =( (parentEnd - parentStart) / 8 );   // size of this atom
        literal longint ='sean';                      // atom type
        longint =1;                                   // atom ID
        integer =0;
        integer = $$CountOf(AtomArray);
        longint =0;
        array AtomArray {
            atomStart:
                // size of this atom
                longint =((atomEnd[$$ArrayIndex(AtomArray)] -
                    atomStart[$$ArrayIndex(AtomArray)]) / 8);
                literal longint;                       // atom type
                longint;                               // atom ID
                integer =0;
                integer =0;                           // no children
                longint =0;
                string;
            atomEnd:
        };
    parentEnd:
};
```

Discussion
Every import component should include a 'thnr' (page 2474) resource holding the same data that MovieImportGetMIMETypeList (page 492) would return. By including this public resource, QuickTime and applications don't need to open the import component and call MovieImportGetMIMETypeList to determine the MIME types the importer supports. In the absence of this resource, QuickTime and applications will use MovieImportGetMIMETypeList. This resource's public type and ID should be 'mime' and 1. Here is an example of such a list:

```
resource 'thnr' (kMyImportComponentResID)  {
    'mime', 1, 0,
    'mime', kMyImportMIMETypeListResID, 0
}
```

Version Notes
Introduced in QuickTime 6.

Programming Info
Resource accessibility: Can be made public
Rez source file: QuicktimeComponents.r
Programming summary: Component Public Resources

'pcki'

Lists streaming payload media supported by a packetizer component.

```
type 'pcki' {
    array infoArray {
        align long;
        hex longint    mediaType;
        hex longint    dataFormat;
        hex longint    vendor;
        hex longint    capabilityFlags;
        byte           canPackMatrixType;
        byte =0;
        byte =0;
        byte =0;
        longint =$$CountOf(characteristicArray);    // array size
        array characteristicArray {
            hex longint    tag;
            hex longint    value;
        };
        hex longint    payloadFlags;
        byte           payloadID;                     // if static payload
        byte =0;
        byte =0;
        byte =0;
        cstring;
    };
};
```

Discussion

Every packetizer must provide a public resource of type `'pcki'`, which contains information about its capabilities. This information lists the `media` types and compression formats the packetizer can work with. It also lists the track characteristics the packetizer can work with, such as layers or transformation matrices. In addition, it provides information about the packetizer's performance characteristics, such as its speed or ability to recover from packet loss. QuickTime selects the packetizer best suited to a stream's current media, compression format, and track characteristics. If there are multiple packetizers that can work with a given track, QuickTime picks the one with the best performance.

Version Notes

Introduced in QuickTime 6.

Programming Info

Resource accessibility: Can be made public
Rez source file: `QTStreamingComponents.r`
Programming summary: `Component Public Resources`

'qter'

Stores error messages for `QTAddMovieError` (page 1420).

```
type 'qter' {
    longint =$$Countof(ErrorSpec);
    wide array ErrorSpec {
    longint; // error code used to find this error
    longint // error type
        kQuickTimeErrorNotice =1,
        kQuickTimeErrorWarning =2,
        kQuickTimeErrorError =3;
    // In the following strings, ^FILENAME, ^APPNAME, ^0, ^1, etc will be
    // replaced as appropriate.
    pstring; // main error string
    pstring; // explanation error string
    pstring; // technical string (not displayed
             // to user except in debug cases)
    align long;
    };
};
```

Version Notes
Introduced in QuickTime 6.

Programming Info
Resource accessibility: Can be made public
Rez source file: `Movies.r`

'rsmi'

Lists the characteristics of streaming payloads supported by a reassembler component.

```
type 'rsmi' {
    array infoArray {
        align long;
        longint =$$CountOf(characteristicArray);    // array size
        array characteristicArray {
            hex longint    tag;
            hex longint    value;
        };
        hex longint    payloadFlags;
        byte           payloadID;    // if static payload
        byte =0;
        byte =0;
        byte =0;
        cstring;
    };
};
```

Discussion
Every reassembler must provide a public resource of type `'rsmi'`, which contains information about its capabilities. This information lists the RTP payload types the reassembler can work with, as well as the reassembler's speed and ability to recover from lost packets. If more than one reassembler is available for a given RTP payload type, QuickTime chooses the one with the best performance characteristics, such as highest speed or best ability to deal with packet loss.

Version Notes
Introduced in QuickTime 6.

Programming Info
Resource accessibility: Can be made public
Rez source file: `QTStreamingComponents.r`
Programming summary: `Component Public Resources`

'skcr'

Defines a media skin content region.

```
// no declaration
```

Discussion
The content of this resource is currently a 1-bit `'pict'` image.

Version Notes
Introduced in QuickTime 6.

Programming Info
Resource accessibility: Can be made public
Programming summary: `Component Public Resources`

'skgr'

Defines a media skin drag region.

```
// no declaration
```

Discussion
The content of this resource is currently a 1-bit `'pict'` image.

Version Notes
Introduced in QuickTime 6.

Programming Info
Resource accessibility: Can be made public
Programming summary: `Component Public Resources`

'snd '

Lists sound commands supported by a sound component.

```
type 'snd ' {
    switch {
        case FormatOne:
            key unsigned integer =$0001;
            unsigned integer =$$CountOf(Synthesizers);
            wide array Synthesizers {
                // Resource ID of synthesizer/modifer
                integer    squareWaveSynth    =$0001,
                           waveTableSynth     =$0003,
                           sampledSynth       =$0005;
                longint;                            // init parameter
                };
        case FormatTwo:
            key unsigned integer =$0002;
            integer free =0, keepInMemory =256+1; // Space for refe count
        };
        unsigned integer =$$CountOf(SoundCmnds);
        wide array SoundCmnds {
            boolean     noData, hasData;
            switch {
                case nullCmd:
                    key bitstring[15] =0;
                    fill word;                      // Param 1 =nil
                    fill long;                      // Param 2 =nil
                case quietCmd:
                    key bitstring[15] =3;
                    fill word;                      // Param 1 =nil
                    fill long;                      // Param 2 =nil
                case flushCmd:
                    key bitstring[15] =4;
                    fill word;                      // Param 1 =nil
                    fill long;                      // Param 2 =nil
                case waitCmd:
                    key bitstring[15] =10;
                    integer    oneSecond =2000;    // Duration
                    fill long;                      // Param 2 =nil
                case pauseCmd:
                    key bitstring[15] =11;
                    fill word;                      // Param 1 =nil
                    fill long;                      // Param 2 =nil
                case resumeCmd:
                    key bitstring[15] =12;
                    fill word;                      // Param 1 =nil
                    fill long;                      // Param 2 =nil
                case callBackCmd:
                    key bitstring[15] =13;
                    integer;                        // User-defined
                    longint;                        // User-defined
                case syncCmd:
                    key bitstring[15] =14;
                    integer;                        // Count
                    longint;                        // Identifier
                case emptyCmd:
                    key bitstring[15] =15;
                    fill word;                      // Param 1 =nil
                    fill long;                      // Param 2 =nil
                case freqDurationCmd:
                    key bitstring[15] =40;
                    integer    oneSecond =2000;    // Duration
                    longint;                        // Frequency
                case restCmd:
                    key bitstring[15] =41;
                    integer    oneSecond =2000;    // Duration
                    fill long;                      // Param 2 =nil
                case freqCmd:
                    key bitstring[15] =42;
                    fill word;                      // Param 1 =nil
                    longint;                        // Frequency
                case ampCmd:
                    key bitstring[15] =43;
                    integer;                        // Amplitude
                    fill long;                      // Param 2
                case timbreCmd:
                    key bitstring[15] =44;
                    integer sineWave, squareWave =255;     // Timbre
                    fill long;                      // Param 2
                case waveTableCmd:
                    key bitstring[15] =60;
                    unsigned integer;               // Length
                    longint;                        // Pointer to table
                case phaseCmd:
                    key bitstring[15] =61;
                    integer;                        // Shift
                    longint;                        // chanPtr
                case soundCmd:
                    key bitstring[15] =80;
                    fill word;                      // Param 1 =nil
                    longint;                        // Pointer to sound
                case bufferCmd:
                    key bitstring[15] =81;
                    fill word;                      // Param 1 =nil
                    longint;                        // Pointer to buffer
                case rateCmd:
                    key bitstring[15] =82;
                    fill word;                      // Param 1 =nil
                    longint;                        // Rate
            };
        };
        array DataTables {
            DataTable:
                fill long;                          // Pointer to data
            SampleCnt:
                unsigned longint;                   // # of sound samples
                unsigned hex longint Rate22K =$56EE8BA3;  // Sampling rate
                unsigned longint;                   // Start of loop
                unsigned longint;                   // End of loop
                hex byte;                           // encode (header type)
                hex byte;                           // baseFrequency
                hex string [$$Long(SampleCnt[$$ArrayIndex(DataTables)])];
        };
};
```

Version Notes
Introduced in QuickTime 6.

Programming Info
Resource accessibility: Private
Rez source file: `Sound.r`
Programming summary: `Component Public Resources`

'src#'

Lists a movie exporter component's supported media types and the minimum and maximum number of sources for each.

```
type 'src#' {
    longint =$$CountOf(SourceArray);
    longint =0;                     // reserved
    array SourceArray {
        literal longint;            // Media type of source
        // min number of sources of this kind required; 0 if none required
        integer;
        // max number of sources of this kind allowed;
        //   65535 if unlimited allowed
        integer;
        longint    isMediaType              =0x01,
                   isMediaCharacteristic    =0x02,
                   isSourceType             =0x04;
    };
};
```

Version Notes
Introduced in QuickTime 6.

Programming Info
Resource accessibility: Can be made public
Rez source file: `QuicktimeComponents.r`
Programming summary: `Component Public Resources`

'stg#'

Lists QuickTime's presets.

```
type 'stg#' {
    hex longint;                    // flags
    longint =$$CountOf(PresetDescriptionArray);
    longint =0;
    array PresetDescriptionArray {
        literal longint;            //  preset key ID
        unsigned hex longint noFlags =0,
                             kQTPresetInfoIsDivider =1;   //  preset flags
        literal longint;            //  preset resource type
        integer;                    //  preset resource ID
        integer =0;                //  padding but also reserved
        integer;                    //  preset name string list ID
        integer;                    //  preset name string index
        integer;                    //  preset description string list ID
        integer;                    //  preset description string index
    };
};
```

Version Notes
Introduced in QuickTime 6.

Programming Info
Resource accessibility: Private
Rez source file: `QuicktimeComponents.r`
Programming summary: `Component Public Resources`

'stgp'

Lists QuickTime's preset platforms.

```
type 'stgp' {
    longint =0;                     // reserved
    literal longint;                 // default settings list resource type
    integer;                         // default settings list resource id
    integer =$$CountOf(SettingsPlatformInfo);
    wide array SettingsPlatformInfo {
        unsigned hex longint =0;  // reserved
        literal longint;           // platform settings list resource Type
        integer;                   // platform settings list resource ID
        // platform type (response from gestaltSysArchitecture)
        integer    platform68k =1,
                   platformPowerPC =2,
                   platformInterpreted =3,
                   platformWin32 =4;
    };
};
```

Version Notes
Introduced in QuickTime 6.

Programming Info
Resource accessibility: Private
Rez source file: `QuickTimeComponents.r`
Programming summary: `Component Public Resources`

'stri'

Contains a component information string.

```
type 'stri' {
     pstring;    // string
};
```

Version Notes
Introduced in QuickTime 6.

Programming Info
Resource accessibility: Private
Rez source file: `Components.r`
Programming summary: `Component Public Resources`

'strn'

Contains a component name string.

```
type 'strn' {
     pstring;    // string
};
```

Version Notes
Introduced in QuickTime 6.

Programming Info
Resource accessibility: Private
Rez source file: `Components.r`
Programming summary: `Component Public Resources`

'sttg'

Lists QuickTime's presets.

```
// no declaration
```

Version Notes
Introduced in QuickTime 6.

Programming Info
Resource accessibility: Private
Programming summary: `Component Public Resources`

'thga'

Lists the characteristics of a component resource alias.

```
type 'thga' {
    literal longint;                                  // type
    literal longint;                                  // subtype
    literal longint;                                  // manufacturer
    unsigned hex longint;                             // component flags
    unsigned hex longint kAnyComponentFlagsMask =0; // component flags mask
    literal longint;                                  // code type
    integer;                                          // code ID
    literal longint;                                  // name type
    integer;                                          // name ID
    literal longint;                                  // info type
    integer;                                          // info ID
    literal longint;                                  // icon type
    integer;                                          // icon ID
    literal longint;                                  // type
    literal longint;                                  // subtype
    literal longint;                                  // manufacturer
    unsigned hex longint;                             // component flags
    unsigned hex longint kAnyComponentFlagsMask =0; // component flags mask
    #if thng_RezTemplateVersion >
=2
        literal longint;                              // resource map type
        integer;                                      // resource map id
        integer    cmpAliasNoFlags =0,
                   cmpAliasOnlyThisFile =1;          // alias flags
    #endif
};
```

Version Notes
Introduced in QuickTime 6.

Programming Info
Resource accessibility: Private
Rez source file: `Components.r`
Programming summary: `Component Public Resources`

'thn#'

Lists a component's load order dependencies.

```
type 'thn#' {
    array {
        literal    longint;    // code type
        integer;               // code ID
    };
};
```

Version Notes
Introduced in QuickTime 6.

Programming Info
Resource accessibility: Private
Rez source file: `Components.r`
Programming summary: `Component Public Resources`

'thnd'

Lists a component's dependencies.

```
type 'thnd' {
    longint =$$CountOf(ComponentDependency);
    wide array ComponentDependency {
        literal longint;         // type
        literal longint;         // subtype
        literal longint;         // manufacturer
        unsigned hex longint;    // component flags
        unsigned hex longint    kAnyComponentFlagsMask =0; // flags mask
    };
};
```

Version Notes
Introduced in QuickTime 6.

Programming Info
Resource accessibility: Private
Rez source file: `Components.r`
Programming summary: `Component Public Resources`

'thng'

Lists the characteristics of a component resource.

```
type 'thng' {
    literal      longint;                          // type
    literal      longint;                          // subtype
    literal      longint;                          // manufacturer
    unsigned hex longint;                          // component flags
    unsigned hex longint kAnyComponentFlagsMask =0; // component flags mask
    literal      longint;                          // code type
    integer;                                       // code ID
    literal      longint;                          // name type
    integer;                                       // name ID
    literal      longint;                          // info type
    integer;                                       // info ID
    literal      longint;                          // icon type
    integer;                                       // icon ID
    #if thng_RezTemplateVersion >
=1
        unsigned hex longint;                      // version
        longint;                                   // registration flags
        integer;                           // resource ID of icon family
        longint =$$CountOf(ComponentPlatformInfo);
        wide array ComponentPlatformInfo {
            unsigned hex    longint;               // component flags
            literal         longint;               // code type
            integer;                               // code ID
            integer    platform68k =1,            // platform type
                       platformPowerPC =2,
                       platformInterpreted =3,
                       platformWin32 =4,
                       platformPowerPCNativeEntryPoint =5;
        };
        #if thng_RezTemplateVersion >
=2
            literal longint;                       // resource map type
            integer;                               // resource map ID
        #endif
    #endif
};
```

Fields

`platform type`

Discussion

The response from `gestaltComponentPlatform` if available, or else from `gestaltSysArchitecture`.

`resource map`

Discussion

See the `'thnr'` (page 2474) resource type.

Discussion

To associate a Public Resource Map with a component, the component's `'thng'` resource must be extended to include a references to a `'thnr'` (page 2474) resource. This can be done when the value of `thng_RezTemplateVersion` is 2, by adding the `resource` type `'thnr'` and an ID. Here is an example:

```
resource 'thng' (512) {
    // component type, subtype, manufacturer, etc. go here
    'thnr', 512
};
```

Version Notes
Introduced in QuickTime 6.

Programming Info
Resource accessibility: Private
Rez source file: `Components.r`
Programming summary: `Component Public Resources`

'thnr'

Contains a public resource map for a component.

```
type 'thnr' {
    array {
        literal longint;        // resource type
        integer;                // resource id
        integer;                // unused flags
        literal    longint;     // Mac OS resource type
        integer;                // Mac OS resource ID
        integer    cmpResourceNoFlags =0,
                   cmpResourceCallComponent =1;    // flags
    };
};
```

Fields
`flags`

Discussion
Some components may need to build the contents of their public resources at run time. For example, the ColorSync visual effect's parameter list varies depending on what color matching methods are installed. In this case, its public component resource cannot be stored in its resource file, but instead must be dynamically created at run time. To indicate that a public resource cannot be loaded directly from a component's file, the component's `'thnr'` resource contains 0 in the `ID` field for that resource and the `cmpResourceCallComponent` flag is set to 1.

Discussion
Public resources are identified by a four-character `OSType` codes and ID numbers. Unlike private resources, however, a public resource's `OSType` code and ID do not have to be the same as the Mac OS resource type and ID that the resource is stored in. Consequently, a component that provides public resources must add a Public Resource Map to the component's `'thng'` (page 2472) resource, giving the mapping between each public resource type and ID and the corresponding private resource type and ID. Here's an example of a Public Resource Map. It makes available two public resource, `'PICT'` 1 and `'PICT'` 2, which are stored in the component as Mac OS resources `'pict'` 128 and `'pict'` 129.

```
resource 'thnr' (512) {
    {
        'PICT', 1, 0, 'pict', 128, 0,
        'PICT', 2, 0, 'pict', 129, 0,
    }
}
```

Version Notes
Introduced in QuickTime 6.

Programming Info
Resource accessibility: Private
Rez source file: `Components.r`

Programming summary: `Component Public Resources`

See Also

For functions that access public resources, see `GetComponentPublicResource` and `GetComponentPublicResourceList`.

Document Revision History

This table describes the changes to *QuickTime Framework Reference*.

Date	Notes
2006-05-23	First publication of this content as a collection of newly published separate QuickTime reference documents.

Index

A

AbortPrePrerollMovie function 177
AddCallBackToTimeBase function 178
AddClonedTrackToMovie function 1534
AddClonedTrackToMovie Values 1665
AddEmptyTrackToMovie function 1535
AddFilePreview function 586
AddImageDescriptionExtension function 586
addMax constant 433
AddMediaDataRef function 1298
AddMediaSample function 1536
AddMediaSample2 function 1539
AddMediaSampleFromEncodedFrame function 1541
AddMediaSampleReference function 1541
AddMediaSampleReferences function 1543
AddMediaSampleReferences64 function 1544
AddMovieExecuteWiredActionsProc function 1298
AddMovieResource function 1299
AddMovieSelection function 1545
AddMovieToStorage function 1301
addOver constant 433
addPin constant 433
AddSampleTableToMedia function 1547
AddSoundDescriptionExtension function 1302
AddTime function 178
AddTrackReference function 1547
AddUserData function 1302
AddUserDataText function 1303
adMax constant 433
adMin constant 433
AlignScreenRect function 587
AlignWindow function 588
Arithmetic and Logical Operator IDs 408
Atom ID Codes 363
AtomicInstrument data type 1763
AtomicInstrumentPtr data type 1763
AttachMovieToCurrentThread function 1304
AttachTimeBaseToCurrentThread function 179
AudioMediaCharacteristic constant 1523

B

badCallOrderErr constant 428
badDepthErr constant 428
BeginFullScreen function 1305
BeginMediaEdits function 1549
blend constant 433

C

CallComponentExecuteWiredAction function 1093
CallMeWhen function 179
CancelCallBack function 180
CanQuickTimeOpenDataRef function 1307
CanQuickTimeOpenFile function 1309
CanQuickTimeOpenFile Values 1521
cantCreateSingleForkFile constant 427
CDSequenceBusy function 589
CDSequenceChangedSourceData function 589
CDSequenceDataSource structure 933
CDSequenceDataSourcePtr data type 936
CDSequenceDisposeDataSource function 590
CDSequenceDisposeMemory function 590
CDSequenceEnd function 591
CDSequenceEquivalentImageDescription function 591
CDSequenceEquivalentImageDescriptionS function 592
CDSequenceFlush function 593
CDSequenceGetDataSource function 594
CDSequenceInvalidate function 594
CDSequenceNewDataSource function 595
CDSequenceNewMemory function 597
CDSequenceSetSourceData function 598
CDSequenceSetSourceDataQueue function 599
CDSequenceSetTimeBase function 599
channelPlayAllData 2245
CheckQuickTimeRegistration function 181
ChooseMovieClock function 181
ClearMovieChanged function 1310
ClearMovieSelection function 1550

ClearMoviesStickyError function 182
ClockCallMeWhen function 453
ClockCancelCallBack function 454
ClockDisposeCallBack function 455
ClockGetRate function 456
ClockGetRateChangeConstraints function 456
ClockGetTime function 457
ClockNewCallBack function 457
ClockRateChanged function 458
ClockSetTimeBase function 459
ClockStartStopChanged function 459
ClockTimeChanged function 460
CloseMovieFile function 1311
CloseMovieStorage function 1312
Codec Flags 373
Codec Identifiers 408
Codec Properties 409, 957
Codec Type Constants 410
codecCompletionDecoded constant 376
codecCompletionDest constant 375
codecCompletionDropped constant 376
codecCompletionForceChainFlush constant 375
codecCompletionNotDisplayable constant 376
codecCompletionNotDrawn constant 376
codecCompletionSource constant 375
codecCompletionUnlockBits constant 375
codecCompletionWentOffscreen constant 375
CodecComponent data type 740
CodecCompressParams structure 936
codecConditionCatchUpDiff constant 960
codecConditionCodecChangedMask constant 961
codecConditionDoCursor constant 960
codecConditionFirstBand constant 959
codecConditionFirstFrame constant 959
codecConditionFirstScreen constant 960
codecConditionLastBand constant 959
codecConditionMaskMayBeChanged constant 960
codecConditionNewAccuracy constant 960
codecConditionNewClut constant 959
codecConditionNewDepth constant 959
codecConditionNewDestination constant 960
codecConditionNewMatte constant 959
codecConditionNewSrcRect constant 959
codecConditionNewTransferMode constant 959
codecConditionNewTransform constant 959
codecConditionToBuffer constant 960
CodecDecompressParams structure 941
CodecDecompressParams Values 963
codecDrawsHigherQualityScaled constant 962
codecFlagDiffFrame constant 373, 750
codecFlagLiveGrab constant 373, 749
codecFlagSupportDisable constant 373, 750
codecFlagUpdatePrevious constant 372, 749
codecFlagUpdatePreviousComp constant 373, 749
codecFlagUsedImageBuffer constant 376
codecFlagWasCompressed constant 372, 749
codecImageBufferIsInAGPMemory constant 961
codecImageBufferIsInPCIMemory constant 410, 961
codecImageBufferIsOverlaySurface constant 961
codecImageBufferMemoryFlagsValid constant 962
CodecInfo Values 379
codecInfoDepth1 constant 380
codecInfoDepth16 constant 381
codecInfoDepth2 constant 380
codecInfoDepth24 constant 381
codecInfoDepth32 constant 381
codecInfoDepth33 constant 381
codecInfoDepth34 constant 381
codecInfoDepth36 constant 381
codecInfoDepth4 constant 380
codecInfoDepth40 constant 381
codecInfoDepth8 constant 380
codecInfoDoes1 constant 376
codecInfoDoes16 constant 377
codecInfoDoes2 constant 376
codecInfoDoes32 constant 377
codecInfoDoes4 constant 376
codecInfoDoes8 constant 376
codecInfoDoesBlend constant 378
codecInfoDoesDither constant 377
codecInfoDoesDouble constant 377
codecInfoDoesHalf constant 377
codecInfoDoesHorizFlip constant 378
codecInfoDoesLossless constant 381
codecInfoDoesMask constant 377
codecInfoDoesMultiPass constant 378
codecInfoDoesQuad constant 377
codecInfoDoesQuarter constant 378
codecInfoDoesRecompress constant 378
codecInfoDoesReorder constant 378
codecInfoDoesRotate constant 378
codecInfoDoesShrink constant 377
codecInfoDoesSpool constant 379
codecInfoDoesStretch constant 377
codecInfoDoesTemporal constant 377
codecInfoDoesVertFlip constant 378
codecInfoDoesWarp constant 378
codecInfoHasEffectParameterList constant 378
codecInfoResourceType constant 410, 961
codecInfoStoresClut constant 381
codecInterfaceVersion constant 410, 961
codecLosslessQuality constant 379
codecLowQuality constant 379
CodecManagerVersion function 600
codecMaxQuality constant 379
codecMinQuality constant 379

CodecNameSpecList structure 740
CodecNameSpecListPtr data type 741
codecNormalQuality constant 379
codecProgressOpen constant 140
codecProgressUpdatePercent constant 140
codecSrcMustBeImageBuffer constant 961
codecSuggestedBufferSentinel constant 410, 961
codecSupportsOutOfOrderDisplayTimes constant 410, 962
codecSupportsScheduledBackwardsPlaybackWith-DifferenceFrames constant 410
codecUsesOverlaySurface constant 961
Color Constants 412
Color Modes 412, 750
CompAdd function 601
CompCompare function 601
CompDiv function 602
CompFixMul function 603
CompMul function 603
CompMulDiv function 604
CompMulDivTrunc function 604
CompNeg function 605
Component Call Selectors 412
Component Identifiers 421
Component Property IDs and Flags 422
componentDllEntryNotFoundErr constant 427
componentDllLoadErr constant 427
ComponentMPWorkFunctionProc callback 932
ComponentMPWorkFunctionUPP data type 949
compositeIn 2349
CompressImage function 605
CompressPicture function 607
CompressPictureFile function 609
CompressSequenceBegin function 609
CompressSequenceFrame function 612
CompShift function 614
CompSquareRoot function 615
CompSub function 615
ConcatMatrix function 616
ConstComponentListPtr data type 2236
ConstFSSpecPtr data type 842
ConstStrFileNameParam data type 741
ControlPtr data type 351
ControlRef data type 352
ConvertDataRefToMovieDataRef function 1551
ConvertFileToMovieFile function 1552
ConvertImage function 616
ConvertMovieToDataRef function 1553
ConvertMovieToFile function 1555
ConvertTime function 183
ConvertTimeScale function 183
ConvertTimeToClockTime function 184
CopyMatrix function 618
CopyMediaMutableSampleTable function 1557
CopyMediaUserData function 1312
CopyMovieSelection function 1558
CopyMovieSettings function 1558
CopyMovieUserData function 1313
CopyTrackSettings function 1559
CopyTrackUserData function 1314
CopyUserData function 1314
CopyUserData Values 1521
CountImageDescriptionExtensionType function 618
CountUserDataType function 1315
CreateMovieControl function 184
CreateMovieControl Values 357
CreateMovieFile function 1316
CreateMovieFile Values 381, 1516
createMovieFileDontOpenFile constant 382, 1516
CreateMovieStorage function 1318
CreateShortcutMovieFile function 1319
currentIn 2357
CurveAddAtomToVectorStream function 854
CurveAddPathAtomToVectorStream function 855
CurveAddZeroAtomToVectorStream function 856
CurveCountPointsInPath function 856
CurveCreateVectorStream function 857
CurveGetAtomDataFromVectorStream function 858
CurveGetLength function 859
CurveGetNearestPathPoint function 859
CurveGetPathPoint function 860
CurveInsertPointIntoPath function 861
CurveLengthToPoint function 862
CurveNewPath function 863
CurvePathPointToLength function 864
CurveSetPathPoint function 865
CutMovieSelection function 1560

D

Data Handler Flags 1171
DataCodecBeginInterruptSafe function 759
DataCodecComponent data type 843
DataCodecCompress function 760
DataCodecCompressPartial function 761
DataCodecDecompress function 762
DataCodecDecompressPartial function 763
DataCodecEndInterruptSafe function 764
DataCodecGetCompressBufferSize function 765
DataHAddMovie function 766
DataHandlerComponent data type 1661
DataHAppend64 function 766
DataHCanUseDataRef function 767
DataHCloseForRead function 768
DataHCloseForWrite function 769

DataHCompareDataRef function 770
DataHCompletionProc callback 842
DataHCompletionUPP data type 843
DataHCreateFile function 771
DataHCreateFileWithFlags function 772
DataHDeleteFile function 773
DataHDoesBuffer function 773
DataHFileTypeOrderingHandle data type 843
DataHFileTypeOrderingPtr data type 843
DataHFinishData function 774
DataHFlushCache function 775
DataHFlushData function 775
DataHGetAvailableFileSize function 776
DataHGetCacheSizeLimit function 776
DataHGetData function 777
DataHGetDataAvailability function 778
DataHGetDataInBuffer function 779
DataHGetDataRate function 779
DataHGetDataRef function 780
DataHGetDataRefAsType function 780
DataHGetDataRefExtension function 781
DataHGetDataRefWithAnchor function 782
DataHGetDeviceIndex function 782
DataHGetFileName function 783
DataHGetFileSize function 784
DataHGetFileSize64 function 785
DataHGetFileSizeAsync function 785
DataHGetFileTypeOrdering function 786
DataHGetFileTypeOrdering Values 847
DataHGetFreeSpace function 787
DataHGetFreeSpace64 function 787
DataHGetInfo function 788
DataHGetInfoFlags function 789
DataHGetInfoFlags Values 848
DataHGetMacOSFileType function 789
DataHGetMIMEType function 790
DataHGetMIMETypeAsync function 790
DataHGetMovie function 791
DataHGetMovieWithFlags function 792
DataHGetPreferredBlockSize function 793
DataHGetScheduleAheadTime function 794
DataHGetTemporaryDataRefCapabilities function 794
DataHGetVolumeList function 795
DataHIsStreamingDataHandler function 796
DataHOpenForRead function 796
DataHOpenForWrite function 797
DataHPlaybackHints function 798
DataHPlaybackHints64 function 799
DataHPollRead function 800
DataHPreextend function 801
DataHPreextend64 function 801
DataHPutData function 802
DataHReadAsync function 803
DataHRenameFile function 804
DataHResolveDataRef function 805
DataHScheduleData function 806
DataHScheduleData64 function 807
DataHSchedulePtr data type 843
DataHScheduleRecord structure 844
DataHScheduleRecord Values 847
DataHSetCacheSizeLimit function 808
DataHSetDataRef function 809
DataHSetDataRefExtension function 810
DataHSetDataRefWithAnchor function 811
DataHSetFileSize function 811
DataHSetFileSize64 function 812
DataHSetIdleManager function 813
DataHSetMacOSFileType function 813
DataHSetMovieUsageFlags function 814
DataHSetMovieUsageFlags Values 848
DataHSetTimeBase function 814
DataHSetTimeHints function 815
DataHTask function 816
DataHUpdateMovie function 816
DataHUseTemporaryDataRef function 817
DataHVolumeList data type 845
DataHVolumeListPtr data type 845
DataHWrite function 817
DataHWrite64 function 819
DataRateParams structure 741
DataRateParamsPtr data type 742
DecompressImage function 619
DecompressorComponent data type 743
DecompressSequenceBegin function 621
DecompressSequenceBeginS function 622
DecompressSequenceFrame function 624
DecompressSequenceFrameS function 625
DecompressSequenceFrameWhen function 626
DeleteMovieFile function 1320
DeleteMovieSegment function 1561
DeleteMovieStorage function 1321
DeleteTrackReference function 1562
DeleteTrackSegment function 1563
DetachMovieFromCurrentThread function 1321
DetachTimeBaseFromCurrentThread function 186
dfClipToTextBox constant 407
dfDontAutoScale constant 407
dfDontDisplay constant 407
dfHorizScroll constant 407
dfReverseScroll constant 407
dfScrollIn constant 407
dfScrollOut constant 407
dfShrinkTextBoxToFit constant 407
dfUseMovieBGColor constant 407
digiInDoesBW constant 2353

digiInDoesColor constant 2353
digiInDoesComponent constant 2353
digiInDoesComposite constant 2352
digiInDoesGenLock constant 2352
digiInDoesNTSC constant 2352
digiInDoesPAL constant 2352
digiInDoesSECAM constant 2352
digiInSignalLock constant 2353
digiInVTR_Broadcast constant 2353
digiOutDoes1 constant 2353
digiOutDoes16 constant 2354
digiOutDoes2 constant 2353
digiOutDoes32 constant 2354
digiOutDoes4 constant 2353
digiOutDoes8 constant 2354
digiOutDoesAsyncGrabs constant 2357
digiOutDoesBlend constant 2356
digiOutDoesCompress constant 2357
digiOutDoesCompressOnly constant 2357
digiOutDoesDither constant 2354
digiOutDoesDouble constant 2355
digiOutDoesHorizFlip constant 2356
digiOutDoesHWPlayThru constant 2356
digiOutDoesILUT constant 2357
digiOutDoesKeyColor constant 2357
digiOutDoesMask constant 2354
digiOutDoesPlayThruDuringCompress constant 2357
digiOutDoesQuad constant 2355
digiOutDoesQuarter constant 2355
digiOutDoesRotate constant 2355
digiOutDoesShrink constant 2354
digiOutDoesSixteenth constant 2355
digiOutDoesSkew constant 2356
digiOutDoesStretch constant 2354
digiOutDoesUnreadableScreenBits constant 2357
digiOutDoesVertFlip constant 2356
digiOutDoesWarp constant 2356
DigitizerInfo structure 2340
digiUnimpErr constant 427
DisposeActionsUPP function 1322
DisposeAllSprites function 1322
DisposeCallBack function 186
DisposeCDataHandlerUPP function 820
DisposeCharDataHandlerUPP function 820
DisposeCodecNameList function 628
DisposeCommentHandlerUPP function 821
DisposeDataHCompletionUPP function 821
DisposeDoMCActionUPP function 1323
DisposeEndDocumentHandlerUPP function 822
DisposeEndElementHandlerUPP function 822
DisposeGetMovieUPP function 1323
DisposeICMAlignmentUPP function 21
DisposeICMCompletionUPP function 22
DisposeICMConvertDataFormatUPP function 22
DisposeICMCursorShieldedUPP function 23
DisposeICMDataUPP function 23
DisposeICMFlushUPP function 24
DisposeICMMemoryDisposedUPP function 24
DisposeICMProgressUPP function 25
DisposeImageCodecDrawBandCompleteUPP function 866
DisposeImageCodecMPDrawBandUPP function 866
DisposeImageCodecTimeTriggerUPP function 867
DisposeMatte function 187
DisposeMCActionFilterUPP function 1179
DisposeMCActionFilterWithRefConUPP function 1179
DisposeMovie function 188
DisposeMovieController function 1324
DisposeMovieDrawingCompleteUPP function 1326
DisposeMovieEditState function 1564
DisposeMovieExecuteWiredActionsUPP function 1327
DisposeMovieExportGetDataUPP function 461
DisposeMovieExportGetPropertyUPP function 461
DisposeMovieExportStageReachedCallbackUPP function 462
DisposeMoviePrePrerollCompleteUPP function 1327
DisposeMoviePreviewCallOutUPP function 1328
DisposeMovieProgressUPP function 1328
DisposeMovieRgnCoverUPP function 1329
DisposeMoviesErrorUPP function 1329
DisposeMovieTrack function 1564
DisposeMusicMIDISendUPP function (Deprecated in Mac OS X v10.5) 1676
DisposeMusicOfflineDataUPP function (Deprecated in Mac OS X v10.5) 1676
DisposePrePrerollCompleteUPP function 1093
DisposePreprocessInstructionHandlerUPP function 822
DisposeQDPixUPP function 25
DisposeQTBandwidthNotificationUPP function (Deprecated in Mac OS X v10.4) 1232
DisposeQTCallBackUPP function 1330
DisposeQTEffectListFilterUPP function 1330
DisposeQTNextTaskNeededSoonerCallbackUPP function 1331
DisposeQTSModalFilterUPP function 1791
DisposeQTSNotificationUPP function 1791
DisposeQTSPanelFilterUPP function 1792
DisposeQTSyncTaskUPP function 1331
DisposeQTTrackPropertyListenerUPP function 1233
DisposeQTVRBackBufferImagingUPP function 1975
DisposeQTVREnteringNodeUPP function 1976
DisposeQTVRImagingCompleteUPP function 1976
DisposeQTVRInterceptUPP function 1977

DisposeQTVRLeavingNodeUPP function 1977
DisposeQTVRMouseOverHotSpotUPP function 1978
DisposeRTPMPDataReleaseUPP function 1792
DisposeRTPPBCallbackUPP function 1793
DisposeSCModalFilterUPP function 462
DisposeSCModalHookUPP function 463
DisposeSGAddFrameBottleUPP function 2096
DisposeSGCompressBottleUPP function 2096
DisposeSGCompressCompleteBottleUPP function 2097
DisposeSGDataUPP function 2097
DisposeSGDisplayBottleUPP function 2098
DisposeSGDisplayCompressBottleUPP function 2098
DisposeSGGrabBottleUPP function 2099
DisposeSGGrabCompleteBottleUPP function 2099
DisposeSGGrabCompressCompleteBottleUPP function 2099
DisposeSGModalFilterUPP function 2100
DisposeSGTransferFrameBottleUPP function 2100
DisposeSprite function 1331
DisposeSpriteWorld function 1332
DisposeStartDocumentHandlerUPP function 823
DisposeStartElementHandlerUPP function 823
DisposeStdPixUPP function 26
DisposeTextMediaUPP function 1333
DisposeTimeBase function 189
DisposeTrackEditState function 1566
DisposeTrackMedia function 1566
DisposeTrackTransferUPP function 1334
DisposeTuneCallBackUPP function (Deprecated in Mac OS X v10.5) 1677
DisposeTunePlayCallBackUPP function (Deprecated in Mac OS X v10.5) 1677
DisposeTweenerDataUPP function 1334
DisposeUserData function 1335
DisposeVdigIntUPP function 824
ditherCopy constant 434
DragAlignedGrayRgn function 629
DragAlignedWindow function 630
DrawPictureFile function 633
DrawTrimmedPicture function 634
DrawTrimmedPictureFile function 635
DSequence Flags 748

E

EffectsFrameParams structure 949
EffectsFrameParamsPtr data type 950
EffectSource structure 951
EffectSource Values 963
EffectSourcePtr data type 951
EndFullScreen function 1335
EndMediaEdits function 1567
EnterMovies function 189
EnterMoviesOnThread function 191
EnterMoviesOnThread Values 358
EqualMatrix function 636
Error Codes 422
ExecuteCallBack function 192
ExitMovies function 192
ExitMoviesOnThread function 194
ExtendMediaDecodeDurationToDisplayEndTime function 1568

F

FCompressImage function 637
FCompressImage Values 372, 748
FCompressPicture function 639
FCompressPictureFile function 641
FDecompressImage function 643
File Types and Creators 428
FindCodec function 645
FixedRect structure 743
FixExp2 function 646
FixLog2 function 647
FixMulDiv function 647
FixPow function 648
FlashMediaDoButtonActions function 194
FlashMediaFrameLabelToMovieTime function 195
FlashMediaFrameNumberToMovieTime function 196
FlashMediaGetDisplayedFrameNumber function 196
FlashMediaGetFlashVariable function 197
FlashMediaGetRefConBounds function 198
FlashMediaGetRefConID function 198
FlashMediaGetSupportedSwfVersion function 199
FlashMediaIDToRefCon function 200
FlashMediaSetFlashVariable function 200
FlashMediaSetPan function 201
FlashMediaSetZoom function 202
FlashMediaSetZoomRect function 202
FlattenMovie function 1336
FlattenMovieData function 1338
FlattenMovieData Values 382
FlattenMovieDataToDataRef function 1340
FourCharCode data type 1509
FracSinCos function 648
Fract data type 743
FSSpecPtr data type 1509
Full Screen Flags 1517
fullScreenAllowEvents constant 1518
fullScreenCaptureDisplay constant 1518
fullScreenDontChangeMenuBar constant 1518
fullScreenHideCursor constant 1518

fullScreenPreflightSize constant 1518

G

GCPart structure 1764
GDGetScale function 649
GDHasScale function 649
GDSetScale function 650
Generic Music Constants 1777
GenericKnobDescription structure 1765
GenericKnobDescriptionListHandle data type 1765
GenericKnobDescriptionListPtr data type 1766
GetBestDeviceRect function 651
GetCallBackTimeBase function 203
GetCallBackType function 203
GetCodecInfo function 652
GetCodecNameList function 652
GetCompressedImageSize function 653
GetCompressedPixMapInfo function 654
GetCompressionTime function 655
GetCSequenceDataRateParams function 657
GetCSequenceFrameNumber function 657
GetCSequenceKeyFrameRate function 658
GetCSequenceMaxCompressionSize function 658
GetCSequencePrevBuffer function 659
GetDataHandler function 1569
GetDSequenceImageBuffer function 660
GetDSequenceMatrix function 661
GetDSequenceNonScheduledDisplayDirection function 661
GetDSequenceNonScheduledDisplayTime function 662
GetDSequenceScreenBuffer function 662
GetFirstCallBack function 204
GetGraphicsImporterForDataRef function 663
GetGraphicsImporterForDataRefWithFlags function 664
GetGraphicsImporterForFile function 665
GetGraphicsImporterForFileWithFlags function 666
GetGraphicsImporterForFileWithFlags Values 752
GetImageDescriptionCTable function 667
GetImageDescriptionExtension function 668
GetMatrixType function 669
GetMaxCompressionSize function 670
GetMaxLoadedTimeInMovie function 1342
GetMediaAdvanceDecodeTime function 1570
GetMediaCreationTime function 1570
GetMediaDataHandler function 1571
GetMediaDataHandlerDescription function 1571
GetMediaDataRef function 1342
GetMediaDataRef Values 1517
GetMediaDataRefCount function 1344
GetMediaDataSize function 1572
GetMediaDataSize64 function 1573
GetMediaDataSizeTime64 function 1573
GetMediaDecodeDuration function 1574
GetMediaDisplayDuration function 1575
GetMediaDisplayEndTime function 1575
GetMediaDisplayStartTime function 1576
GetMediaDuration function 1576
GetMediaHandler function 1577
GetMediaHandlerDescription function 1578
GetMediaInputMap function 1579
GetMediaLanguage function 1581
GetMediaModificationTime function 1581
GetMediaNextInterestingDecodeTime function 1344
GetMediaNextInterestingDisplayTime function 1345
GetMediaNextInterestingTime function 1346
GetMediaPlayHints function 1348
GetMediaPreferredChunkSize function 1582
GetMediaPropertyAtom function 1348
GetMediaQuality function 1582
GetMediaSample function 1583
GetMediaSample2 function 1585
GetMediaSampleCount function 1586
GetMediaSampleDescription function 1587
GetMediaSampleDescriptionCount function 1588
GetMediaSampleReference function 1589
GetMediaSampleReferences function 1590
GetMediaSampleReferences64 function 1592
GetMediaShadowSync function 1593
GetMediaSyncSampleCount function 1594
GetMediaTimeScale function 1594
GetMediaTrack function 1595
GetMediaUserData function 1595
GetMovieActive function 204
GetMovieActiveSegment function 205
GetMovieAnchorDataRef function 1349
GetMovieAudioBalance function 1350
GetMovieAudioContext function 205
GetMovieAudioFrequencyLevels function 1351
GetMovieAudioFrequencyMeteringBandFrequencies function 1351
GetMovieAudioFrequencyMeteringNumBands function 1352
GetMovieAudioGain function 1353
GetMovieAudioMute function 1353
GetMovieAudioVolumeLevels function 1354
GetMovieAudioVolumeMeteringEnabled function 1355
GetMovieBoundsRgn function 206
GetMovieBox function 207
GetMovieClipRgn function 208

GetMovieColorTable function 1356
GetMovieCompleteParams structure 1164
GetMovieCoverProcs function 1356
GetMovieCreationTime function 209
GetMovieDataSize function 1596
GetMovieDataSize64 function 1596
GetMovieDefaultDataRef function 1357
GetMovieDisplayBoundsRgn function 209
GetMovieDisplayClipRgn function 210
GetMovieDuration function 211
GetMovieGWorld function 212
GetMovieImporter Flags 1665
GetMovieImporterForDataRef function 1597
GetMovieIndTrack function 1598
GetMovieIndTrackType function 1600
GetMovieIndTrackType Values 1665
GetMovieLoadState function 1358
GetMovieMatrix function 213
GetMovieModificationTime function 213
GetMovieNaturalBoundsRect function 214
GetMovieNextInterestingTime function 1359
GetMoviePict function 214
GetMoviePosterPict function 215
GetMoviePosterTime function 216
GetMoviePreferredRate function 217
GetMoviePreferredVolume function 217
GetMoviePreviewMode function 218
GetMoviePreviewTime function 218
GetMovieProc callback 1504
GetMovieProgressProc function 1361
GetMoviePropertyAtom function 1361
GetMovieRate function 219
GetMovieRateChangeConstraints function 220
GetMovieSegmentDisplayBoundsRgn function 1362
GetMovieSelection function 220
GetMoviesError function 221
GetMoviesStickyError function 222
GetMovieStatus function 1363
GetMovieThreadAttachState function 1363
GetMovieTime function 223
GetMovieTimeBase function 224
GetMovieTimeScale function 224
GetMovieTrack function 1601
GetMovieTrackCount function 1601
GetMovieUPP data type 1509
GetMovieUserData function 225
GetMovieVisualBrightness function 1364
GetMovieVisualContext function 226
GetMovieVisualContrast function 1364
GetMovieVisualHue function 1365
GetMovieVisualSaturation function 1366
GetMovieVolume function 226
GetNextCallBack function 227
GetNextImageDescriptionExtensionType function 672
GetNextTrackForCompositing function 227
GetNextTrackReferenceType function 1602
GetNextUserDataType function 1366
GetPictureFileHeader function 673
GetPosterBox function 1367
GetPrevTrackForCompositing function 228
GetQuickTimePreference function 1368
GetSimilarity function 673
GetSoundDescriptionExtension function 1369
GetSpriteProperty function 1370
GetTimeBaseEffectiveRate function 228
GetTimeBaseFlags function 229
GetTimeBaseMasterClock function 229
GetTimeBaseMasterOffsetTimeBase function 230
GetTimeBaseMasterTimeBase function 231
GetTimeBaseRate function 231
GetTimeBaseRateChangeStatus function 232
GetTimeBaseStartTime function 233
GetTimeBaseStatus function 234
GetTimeBaseStopTime function 234
GetTimeBaseThreadAttachState function 235
GetTimeBaseTime function 235
GetTrackAlternate function 1603
GetTrackAudioGain function 1370
GetTrackAudioMute function 1371
GetTrackBoundsRgn function 236
GetTrackClipRgn function 237
GetTrackCreationTime function 1604
GetTrackDataSize function 1604
GetTrackDataSize64 function 1605
GetTrackDimensions function 1606
GetTrackDisplayBoundsRgn function 237
GetTrackDisplayMatrix function 1607
GetTrackDuration function 1607
GetTrackEditRate function 1608
GetTrackEditRate64 function 1609
GetTrackEnabled function 1609
GetTrackID function 1610
GetTrackLayer function 1611
GetTrackLoadSettings function 1372
GetTrackMatrix function 1611
GetTrackMatte function 238
GetTrackMedia function 1612
GetTrackModificationTime function 1613
GetTrackMovie function 1613
GetTrackMovieBoundsRgn function 239
GetTrackNextInterestingTime function 1373
GetTrackOffset function 1614
GetTrackPict function 239
GetTrackReference function 1614
GetTrackReferenceCount function 1615

GetTrackSegmentDisplayBoundsRgn function 1374
GetTrackSoundLocalizationSettings function 1616
GetTrackStatus function 1375
GetTrackUsage function 1616
GetTrackUserData function 1617
GetTrackVolume function 1618
GetUserData function 1375
GetUserDataItem function 1376
GetUserDataText function 1377
GoToBeginningOfMovie function 240
GoToEndOfMovie function 241
GrafPort structure 2344
GrafPtr data type 2347
GraphicImageMovieImportComponent data type 555
Graphics Importer Flags 1082
Graphics Transfer Modes 430
GraphicsExportCanTranscode function 974
GraphicsExportCanUseCompressor function 975
GraphicsExportComponent data type 1081
GraphicsExportDoExport function 975
GraphicsExportDoStandaloneExport function 976
GraphicsExportDoTranscode function 977
GraphicsExportDoUseCompressor function 977
GraphicsExportDrawInputImage function 978
GraphicsExportGetColorSyncProfile function 979
GraphicsExportGetCompressionMethod function 980
GraphicsExportGetCompressionQuality function 980
GraphicsExportGetDefaultFileNameExtension function 981
GraphicsExportGetDefaultFileTypeAndCreator function 982
GraphicsExportGetDepth function 982
GraphicsExportGetDontRecompress function 983
GraphicsExportGetExifEnabled function 983
GraphicsExportGetInputCGBitmapContext function 984
GraphicsExportGetInputCGImage function 984
GraphicsExportGetInputDataReference function 985
GraphicsExportGetInputDataSize function 986
GraphicsExportGetInputFile function 986
GraphicsExportGetInputGraphicsImporter function 987
GraphicsExportGetInputGWorld function 988
GraphicsExportGetInputHandle function 988
GraphicsExportGetInputImageDepth function 989
GraphicsExportGetInputImageDescription function 989
GraphicsExportGetInputImageDimensions function 990
GraphicsExportGetInputOffsetAndLimit function 991
GraphicsExportGetInputPicture function 991
GraphicsExportGetInputPixmap function 992
GraphicsExportGetInputPtr function 993
GraphicsExportGetInterlaceStyle function 994
GraphicsExportGetMetaData function 994
GraphicsExportGetMIMETypeList function 995
GraphicsExportGetOutputDataReference function 995
GraphicsExportGetOutputFile function 996
GraphicsExportGetOutputFileTypeAndCreator function 997
GraphicsExportGetOutputHandle function 997
GraphicsExportGetOutputMark function 998
GraphicsExportGetOutputOffsetAndMaxSize function 998
GraphicsExportGetProgressProc function 999
GraphicsExportGetResolution function 1000
GraphicsExportGetSettingsAsAtomContainer function 1000
GraphicsExportGetSettingsAsText function 1001
GraphicsExportGetTargetDataSize function 1001
GraphicsExportGetThumbnailEnabled function 1002
GraphicsExportMayExporterReadInputData function 1002
GraphicsExportReadInputData function 1003
GraphicsExportReadOutputData function 1004
GraphicsExportRequestSettings function 1005
GraphicsExportSetColorSyncProfile function 1006
GraphicsExportSetCompressionMethod function 1006
GraphicsExportSetCompressionQuality function 1007
GraphicsExportSetDepth function 1008
GraphicsExportSetDontRecompress function 1008
GraphicsExportSetExifEnabled function 1009
GraphicsExportSetInputCGBitmapContext function 1010
GraphicsExportSetInputCGImage function 1010
GraphicsExportSetInputDataReference function 1011
GraphicsExportSetInputFile function 1011
GraphicsExportSetInputGraphicsImporter function 1012
GraphicsExportSetInputGWorld function 1013
GraphicsExportSetInputHandle function 1014
GraphicsExportSetInputOffsetAndLimit function 1015
GraphicsExportSetInputPicture function 1015
GraphicsExportSetInputPixmap function 1016
GraphicsExportSetInputPtr function 1017
GraphicsExportSetInterlaceStyle function 1018
GraphicsExportSetMetaData function 1018

GraphicsExportSetOutputDataReference function 1019
GraphicsExportSetOutputFile function 1020
GraphicsExportSetOutputFileTypeAndCreator function 1020
GraphicsExportSetOutputHandle function 1021
GraphicsExportSetOutputMark function 1021
GraphicsExportSetOutputOffsetAndMaxSize function 1022
GraphicsExportSetProgressProc function 1023
GraphicsExportSetResolution function 1023
GraphicsExportSetSettingsFromAtomContainer function 1024
GraphicsExportSetTargetDataSize function 1025
GraphicsExportSetThumbnailEnabled function 1025
GraphicsExportWriteOutputData function 1026
GraphicsImageImportGetSequenceEnabled function 463
GraphicsImageImportSetSequenceEnabled function 464
GraphicsImportComponent data type 1081
GraphicsImportCreateCGImage function 1027
GraphicsImportCreateCGImage Values 1082
GraphicsImportDoesDrawAllPixels function 1027
GraphicsImportDoesDrawAllPixels Values 1081
GraphicsImportDoExportImageFileDialog function 1028
GraphicsImportDoExportImageFileToDataRefDialog function 1029
GraphicsImportDraw function 1030
GraphicsImportExportImageFile function 1031
GraphicsImportExportImageFileToDataRef function 1033
GraphicsImportGetAliasedDataReference function 1033
GraphicsImportGetAsPicture function 1034
GraphicsImportGetBaseDataOffsetAndSize64 function 1035
GraphicsImportGetBoundsRect function 1035
GraphicsImportGetClip function 1036
GraphicsImportGetColorSyncProfile function 1037
GraphicsImportGetDataFile function 1037
GraphicsImportGetDataHandle function 1038
GraphicsImportGetDataOffsetAndSize function 1039
GraphicsImportGetDataOffsetAndSize64 function 1040
GraphicsImportGetDataReference function 1040
GraphicsImportGetDataReferenceOffsetAndLimit function 1041
GraphicsImportGetDataReferenceOffsetAndLimit64 function 1042
GraphicsImportGetDefaultClip function 1043
GraphicsImportGetDefaultGraphicsMode function 1044
GraphicsImportGetDefaultMatrix function 1045
GraphicsImportGetDefaultSourceRect function 1045
GraphicsImportGetDestinationColorSyncProfileRef function 1046
GraphicsImportGetDestRect function 1046
GraphicsImportGetExportImageTypeList function 1047
GraphicsImportGetExportSettingsAsAtomContainer function 1048
GraphicsImportGetFlags function 1049
GraphicsImportGetGenericColorSyncProfile function 1049
GraphicsImportGetGraphicsMode function 1050
GraphicsImportGetGWorld function 1050
GraphicsImportGetImageCount function 1051
GraphicsImportGetImageDescription function 1052
GraphicsImportGetImageIndex function 1053
GraphicsImportGetMatrix function 1053
GraphicsImportGetMetaData function 1054
GraphicsImportGetMIMETypeList function 1055
GraphicsImportGetNaturalBounds function 1055
GraphicsImportGetOverrideSourceColorSyncProfileRef function 1056
GraphicsImportGetProgressProc function 1057
GraphicsImportGetQuality function 1057
GraphicsImportGetSourceRect function 1058
GraphicsImportReadData function 1059
GraphicsImportReadData64 function 1060
GraphicsImportSaveAsPicture function 1060
GraphicsImportSaveAsPictureToDataRef function 1061
GraphicsImportSaveAsQuickTimeImageFile function 1062
GraphicsImportSaveAsQuickTimeImageFileToDataRef function 1063
GraphicsImportSetBoundsRect function 1063
GraphicsImportSetClip function 1064
GraphicsImportSetDataFile function 1065
GraphicsImportSetDataHandle function 1066
GraphicsImportSetDataReference function 1067
GraphicsImportSetDataReferenceOffsetAndLimit function 1068
GraphicsImportSetDataReferenceOffsetAndLimit64 function 1069
GraphicsImportSetDestinationColorSyncProfileRef function 1069
GraphicsImportSetDestRect function 1070
GraphicsImportSetExportSettingsFromAtomContainer function 1070
GraphicsImportSetFlags function 1071

GraphicsImportSetGraphicsMode function 1072
GraphicsImportSetGWorld function 1072
GraphicsImportSetImageIndex function 1073
GraphicsImportSetImageIndexToThumbnail function 1074
GraphicsImportSetMatrix function 1075
GraphicsImportSetOverrideSourceColorSyncProfileRef function 1075
GraphicsImportSetProgressProc function 1076
GraphicsImportSetQuality function 1077
GraphicsImportSetSourceRect function 1078
GraphicsImportValidate function 1078
GraphicsImportWillUseColorMatching function 1079
grayishTextOr constant 432
gxPaths structure 952
gxPoint structure 952

H

HandlerError data type 555
HasMovieChanged function 1378
hilite constant 432
hilitetransfermode constant 432
HIMovieViewChangeAttributes function 1180
HIMovieViewCreate function 1180
HIMovieViewGetAttributes function 1181
HIMovieViewGetControllerBarSize function 1181
HIMovieViewGetMovie function 1182
HIMovieViewGetMovieController function 1182
HIMovieViewPause function 1183
HIMovieViewPlay function 1183
HIMovieViewSetMovie function 1184
Hint Flags 1518
hintsAllowIdleSleep constant 1519
HitTestDSequenceData function 674

I

ICM Preferences and Flags 382
ICM Property IDs 140
ICMAlignmentProc callback 130
ICMAlignmentUPP data type 136
ICMCompletionProc callback 131
ICMCompletionUPP data type 136
ICMCompressionFrameOptionsCreate function 26
ICMCompressionFrameOptionsCreateCopy function 27
ICMCompressionFrameOptionsGetForceKeyFrame function 27
ICMCompressionFrameOptionsGetFrameType function 28
ICMCompressionFrameOptionsGetProperty function 28
ICMCompressionFrameOptionsGetPropertyInfo function 29
ICMCompressionFrameOptionsGetTypeID function 30
ICMCompressionFrameOptionsRelease function 30
ICMCompressionFrameOptionsRetain function 31
ICMCompressionFrameOptionsSetForceKeyFrame function 31
ICMCompressionFrameOptionsSetFrameType function 32
ICMCompressionFrameOptionsSetProperty function 33
ICMCompressionSessionBeginPass function 33
ICMCompressionSessionCompleteFrames function 34
ICMCompressionSessionCreate function 35
ICMCompressionSessionEncodeFrame function 37
ICMCompressionSessionEndPass function 38
ICMCompressionSessionGetImageDescription function 38
ICMCompressionSessionGetPixelBufferPool function 39
ICMCompressionSessionGetProperty function 40
ICMCompressionSessionGetPropertyInfo function 41
ICMCompressionSessionGetTimeScale function 42
ICMCompressionSessionGetTypeID function 42
ICMCompressionSessionOptionsCreate function 42
ICMCompressionSessionOptionsCreateCopy function 43
ICMCompressionSessionOptionsGetAllowFrame-Reordering function 44
ICMCompressionSessionOptionsGetAllowFrameTime-Changes function 44
ICMCompressionSessionOptionsGetAllowTemporal-Compression function 44
ICMCompressionSessionOptionsGetDurationsNeeded function 45
ICMCompressionSessionOptionsGetMaxKeyFrameInterval function 45
ICMCompressionSessionOptionsGetProperty function 46
ICMCompressionSessionOptionsGetPropertyInfo function 47
ICMCompressionSessionOptionsGetTypeID function 48
ICMCompressionSessionOptionsRelease function 49
ICMCompressionSessionOptionsRetain function 49

ICMCompressionSessionOptionsSetAllowFrameReordering function 49
ICMCompressionSessionOptionsSetAllowFrameTimeChanges function 50
ICMCompressionSessionOptionsSetAllowTemporalCompression function 51
ICMCompressionSessionOptionsSetDurationsNeeded function 51
ICMCompressionSessionOptionsSetMaxKeyFrameInterval function 52
ICMCompressionSessionOptionsSetProperty function 53
ICMCompressionSessionProcessBetweenPasses function 54
ICMCompressionSessionRelease function 55
ICMCompressionSessionRetain function 55
ICMCompressionSessionSetProperty function 56
ICMCompressionSessionSupportsMultiPassEncoding function 57
ICMCompressorSessionDropFrame function 57
ICMCompressorSessionEmitEncodedFrame function 58
ICMCompressorSourceFrameGetDisplayNumber function 59
ICMCompressorSourceFrameGetDisplayTimeStampAndDuration function 59
ICMCompressorSourceFrameGetFrameOptions function 60
ICMCompressorSourceFrameGetPixelBuffer function 60
ICMCompressorSourceFrameGetTypeID function 61
ICMCompressorSourceFrameRelease function 61
ICMCompressorSourceFrameRetain function 62
ICMCursorShieldedProc callback 131
ICMCursorShieldedUPP data type 136
ICMDataProc callback 132
ICMDataUPP data type 137
ICMDecompressComplete function 675
ICMDecompressCompleteS function 676
ICMDecompressionFrameOptionsCreate function 62
ICMDecompressionFrameOptionsCreateCopy function 63
ICMDecompressionFrameOptionsGetProperty function 63
ICMDecompressionFrameOptionsGetPropertyInfo function 64
ICMDecompressionFrameOptionsGetTypeID function 65
ICMDecompressionFrameOptionsRelease function 66
ICMDecompressionFrameOptionsRetain function 66
ICMDecompressionFrameOptionsSetProperty function 66
ICMDecompressionSessionCreate function 67
ICMDecompressionSessionCreateForVisualContext function 68
ICMDecompressionSessionDecodeFrame function 69
ICMDecompressionSessionFlush function 70
ICMDecompressionSessionGetProperty function 71
ICMDecompressionSessionGetPropertyInfo function 72
ICMDecompressionSessionGetTypeID function 73
ICMDecompressionSessionOptionsCreate function 73
ICMDecompressionSessionOptionsCreateCopy function 74
ICMDecompressionSessionOptionsGetProperty function 74
ICMDecompressionSessionOptionsGetPropertyInfo function 75
ICMDecompressionSessionOptionsGetTypeID function 76
ICMDecompressionSessionOptionsRelease function 76
ICMDecompressionSessionOptionsRetain function 77
ICMDecompressionSessionOptionsSetProperty function 77
ICMDecompressionSessionRelease function 78
ICMDecompressionSessionRetain function 79
ICMDecompressionSessionSetNonScheduledDisplayDirection function 79
ICMDecompressionSessionSetNonScheduledDisplayTime function 80
ICMDecompressionSessionSetProperty function 81
ICMDecompressionTrackingCallbackRecord structure 137
ICMEncodedFrameCreateMutable function 82
ICMEncodedFrameGetBufferSize function 82
ICMEncodedFrameGetDataPtr function 83
ICMEncodedFrameGetDataSize function 83
ICMEncodedFrameGetDecodeDuration function 84
ICMEncodedFrameGetDecodeNumber function 84
ICMEncodedFrameGetDecodeTimeStamp function 85
ICMEncodedFrameGetDisplayDuration function 85
ICMEncodedFrameGetDisplayOffset function 85
ICMEncodedFrameGetDisplayTimeStamp function 86
ICMEncodedFrameGetFrameType function 86
ICMEncodedFrameGetImageDescription function 87
ICMEncodedFrameGetMediaSampleFlags function 88
ICMEncodedFrameGetSimilarity function 88
ICMEncodedFrameGetSourceFrameRefCon function 88
ICMEncodedFrameGetTimeScale function 89
ICMEncodedFrameGetTypeID function 89
ICMEncodedFrameGetValidTimeFlags function 90

ICMEncodedFrameRelease function 90
ICMEncodedFrameRetain function 91
ICMEncodedFrameSetDataSize function 91
ICMEncodedFrameSetDecodeDuration function 91
ICMEncodedFrameSetDecodeTimeStamp function 92
ICMEncodedFrameSetDisplayDuration function 92
ICMEncodedFrameSetDisplayTimeStamp function 93
ICMEncodedFrameSetFrameType function 93
ICMEncodedFrameSetFrameType Values 160
ICMEncodedFrameSetMediaSampleFlags function 94
ICMEncodedFrameSetSimilarity function 95
ICMEncodedFrameSetValidTimeFlags function 95
ICMFlushProc callback 132
ICMFlushUPP data type 137
icmFrameTimeHasDecodeTime constant 383
icmFrameTimeHasVirtualStartTimeAndDuration constant 383
ICMGetPixelFormatInfo function 677
ICMImageDescriptionGetProperty function 96
ICMImageDescriptionGetPropertyInfo function 97
ICMImageDescriptionSetProperty function 97
ICMMultiPassStorageCallbacks structure 137
ICMMultiPassStorageCopyDataAtTimeStamp function 98
ICMMultiPassStorageCreateWithCallbacks function 99
ICMMultiPassStorageCreateWithTemporaryFile function 99
ICMMultiPassStorageCreateWithTemporaryFile Values 160
ICMMultiPassStorageGetTimeStamp function 100
ICMMultiPassStorageGetTimeStamp Values 160
ICMMultiPassStorageGetTypeID function 101
ICMMultiPassStorageRelease function 101
ICMMultiPassStorageRetain function 102
ICMMultiPassStorageSetDataAtTimeStamp function 102
ICMPixelFormatInfo structure 744
ICMPixelFormatInfoPtr data type 744
ICMProgressProc callback 133
ICMProgressProc Values 140
ICMProgressUPP data type 138
ICMSequenceGetChainMember function 677
ICMSequenceGetInfo function 678
ICMSequenceLockBits function 679
ICMSequenceSetInfo function 679
ICMSequenceUnlockBits function 680
ICMSetPixelFormatInfo function 681
ICMShieldSequenceCursor function 681
identityMatrixType constant 1957
ImageCodecBandCompress function 867
ImageCodecBandDecompress function 868
ImageCodecBeginBand function 869
ImageCodecBeginPass function 870
ImageCodecBusy function 871
ImageCodecCancelTrigger function 872
ImageCodecCompleteFrame function 872
ImageCodecCreateStandardParameterDialog function 873
ImageCodecDecodeBand function 874
ImageCodecDismissStandardParameterDialog function 875
ImageCodecDisposeImageGWorld function 876
ImageCodecDisposeMemory function 876
ImageCodecDITLEvent function 877
ImageCodecDITLInstall function 878
ImageCodecDITLItem function 878
ImageCodecDITLRemove function 879
ImageCodecDITLValidateInput function 880
ImageCodecDrawBand function 880
ImageCodecDroppingFrame function 881
ImageCodecEffectBegin function 882
ImageCodecEffectCancel function 882
ImageCodecEffectConvertEffectSourceToFormat function 883
ImageCodecEffectDisposeSMPTEFrame function 884
ImageCodecEffectGetSpeed function 884
ImageCodecEffectPrepareSMPTEFrame function 885
ImageCodecEffectRenderFrame function 885
ImageCodecEffectRenderSMPTEFrame function 886
ImageCodecEffectSetup function 887
ImageCodecEncodeFrame function 888
ImageCodecEndBand function 888
ImageCodecExtractAndCombineFields function 889
ImageCodecFlush function 891
ImageCodecFlushFrame function 892
ImageCodecGetBaseMPWorkFunction function 892
ImageCodecGetCodecInfo function 893
ImageCodecGetCompressedImageSize function 894
ImageCodecGetCompressionTime function 895
ImageCodecGetDecompressLatency function 896
ImageCodecGetDITLForSize function 897
ImageCodecGetMaxCompressionSize function 898
ImageCodecGetMaxCompressionSizeWithSources function 899
ImageCodecGetParameterList function 900
ImageCodecGetParameterListHandle function 901
ImageCodecGetSettings function 901
ImageCodecGetSettingsAsText function 902
ImageCodecGetSimilarity function 902
ImageCodecGetSourceDataGammaLevel function 903
ImageCodecHitTestData function 904
ImageCodecHitTestDataWithFlags function 905
ImageCodecInitialize function 906
ImageCodecIsImageDescriptionEquivalent function 906

ImageCodecIsStandardParameterDialogEvent function 907
ImageCodecMergeFloatingImageOntoWindow function 908
ImageCodecMPDrawBandProc callback 932
ImageCodecMPDrawBandUPP data type 953
ImageCodecNewImageBufferMemory function 909
ImageCodecNewImageGWorld function 910
ImageCodecNewMemory function 911
ImageCodecPreCompress function 912
ImageCodecPreDecompress function 912
ImageCodecPreflight function 913
ImageCodecPrepareToCompressFrames function 914
ImageCodecProcessBetweenPasses function 915
ImageCodecQueueStarting function 916
ImageCodecQueueStopping function 916
ImageCodecRemoveFloatingImage function 917
ImageCodecRequestGammaLevel function 918
ImageCodecRequestSettings function 919
ImageCodecScheduleFrame function 920
ImageCodecSetSettings function 920
ImageCodecSetTimeBase function 921
ImageCodecSetTimeCode function 922
ImageCodecSourceChanged function 922
ImageCodecStandardParameterDialogDoAction function 923
ImageCodecTimeTriggerProc callback 933
ImageCodecTimeTriggerUPP data type 953
ImageCodecTrimImage function 925
ImageCodecValidateParameters function 926
ImageCodecValidateParameters Values 963
ImageFieldSequence data type 745
ImageFieldSequenceBegin function 682
ImageFieldSequenceEnd function 683
ImageFieldSequenceExtractCombine function 683
ImageFieldSequenceExtractCombine Values 384
ImageSequenceDataSource data type 745
ImageSubCodecDecompressCapabilities structure 954
ImageSubCodecDecompressRecord structure 954
ImageSubCodecDecompressRecord Values 962
ImageTranscodeDisposeFrameData function 685
ImageTranscodeFrame function 685
ImageTranscoderBeginSequence function 103
ImageTranscoderComponent data type 139
ImageTranscoderConvert function 104
ImageTranscoderDisposeData function 105
ImageTranscoderEndSequence function 106
ImageTranscodeSequence data type 745
ImageTranscodeSequenceBegin function 686
ImageTranscodeSequenceEnd function 687
InitializeQTS function 1793
InsertEmptyMovieSegment function 1618
InsertEmptyTrackSegment function 1619
InsertMediaIntoTrack function 1620
InsertMovieSegment function 1622
InsertTrackSegment function 1623
InstrumentAboutInfo structure 1766
InstrumentInfoListHandle data type 1767
InstrumentInfoListPtr data type 1767
InvalidateMovieRegion function 241
InvalidateSprite function 1379
InvalidateSpriteWorld function 1380
InverseMatrix function 687
InvokeQTTrackPropertyListenerUPP function 1233
IsMovieDone function 242
IsScrapMovie function 1624
ITextAddString function 243
ITextGetString function 244
ITextRemoveString function 244
ITextRemoveString Values 357

K

k422YpCbCr10CodecType constant 411
k422YpCbCr16CodecType constant 411
k422YpCbCr8CodecType constant 411
k4444YpCbCrA8CodecType constant 411
k4444YpCbCrA8RCodecType constant 412
k444YpCbCr10CodecType constant 411
k444YpCbCr8CodecType constant 411
kCodecFrameTypeDifference constant 962
kCodecFrameTypeKey constant 962
kCodecFrameTypeUnknown constant 962
kComponentPropertyCacheFlagIsDynamic constant 385
kComponentPropertyCacheFlagNotPersistent constant 385
kComponentPropertyCacheFlags constant 385
kComponentPropertyCacheSeed constant 385
kComponentPropertyClassPropertyInfo constant 385
kComponentPropertyInfoList constant 385
kConnectionActive constant 1959
kDataHCanRead 385, 846
kDataHCanRead constant 386, 846
kDataHCanStreamingWrite constant 386, 847
kDataHCanWrite constant 386, 847
kDataHSpecialRead constant 386, 846
kDataHSpecialReadFile constant 386, 847
kDataHSpecialWrite constant 386, 847
kEffectRawSource constant 963
kGenericMusicAllDefaults constant 1779
kGenericMusicDrumKnob constant 1779

kGenericMusicMiscLongFirstGMDrumHW constant 1779
kGenericMusicMiscLongFirstGMHW constant 1779
kGenericMusicMiscLongFirstUserHW constant 1779
kGenericMusicResAboutPICT constant 1781
kGenericMusicResBitsLongList constant 1780
kGenericMusicResDrumKnobDescriptionList constant 1780
kGenericMusicResDrumList constant 1780
kGenericMusicResGMTranslation constant 1780
kGenericMusicResInstrumentKnobDescriptionList constant 1780
kGenericMusicResInstrumentList constant 1780
kGenericMusicResKnobDescriptionList constant 1780
kGenericMusicResMiscLongList constant 1780
kGenericMusicResMiscStringList constant 1779
kGenericMusicResModifiableInstrumentHW constant 1780
kGenericMusicResROMInstrumentData constant 1780
kICMCompressionFrameOptionsPropertyID_-ForceKeyFrame constant 152
kICMCompressionFrameOptionsPropertyID_FrameType constant 152
kICMCompressionSessionOptionsPropertyID_-AllowAsyncCompletion constant 153
kICMCompressionSessionOptionsPropertyID_-AllowFrameReordering constant 153
kICMCompressionSessionOptionsPropertyID_-AllowFrameTimeChanges constant 153
kICMCompressionSessionOptionsPropertyID_-AllowTemporalCompression constant 152
kICMCompressionSessionOptionsPropertyID_-AverageDataRate constant 155
kICMCompressionSessionOptionsPropertyID_-CleanAperture constant 156
kICMCompressionSessionOptionsPropertyID_ColorTable constant 154
kICMCompressionSessionOptionsPropertyID_-CompressorComponent constant 154
kICMCompressionSessionOptionsPropertyID_-CompressorSettings constant 154
kICMCompressionSessionOptionsPropertyID_-CPUTimeBudget constant 155
kICMCompressionSessionOptionsPropertyID_-DataRateLimitCount constant 155
kICMCompressionSessionOptionsPropertyID_-DataRateLimits constant 155
kICMCompressionSessionOptionsPropertyID_Depth constant 154
kICMCompressionSessionOptionsPropertyID_-DurationsNeeded constant 153
kICMCompressionSessionOptionsPropertyID_-ExpectedFrameRate constant 156
kICMCompressionSessionOptionsPropertyID_FieldInfo constant 156
kICMCompressionSessionOptionsPropertyID_-MaxDataRateLimits constant 155
kICMCompressionSessionOptionsPropertyID_-MaxFrameDelayCount constant 154
kICMCompressionSessionOptionsPropertyID_-MaxFrameDelayTime constant 154
kICMCompressionSessionOptionsPropertyID_-MaxKeyFrameInterval constant 153
kICMCompressionSessionOptionsPropertyID_-MaxPartialSyncFrameInterval constant 153
kICMCompressionSessionOptionsPropertyID_-MultiPassStorage constant 156
kICMCompressionSessionOptionsPropertyID_-PixelAspectRatio constant 156
kICMCompressionSessionOptionsPropertyID_Quality constant 154
kICMCompressionSessionOptionsPropertyID_-ScalingMode constant 156
kICMCompressionSessionOptionsPropertyID_-SourceFrameCount constant 156
kICMCompressionSessionOptionsPropertyID_-WasCompressed constant 155
kICMCompressionSessionPropertyID_-CompressorPixelBufferAttributes constant 157
kICMCompressionSessionPropertyID_ImageDescription constant 157
kICMCompressionSessionPropertyID_PixelBufferPool constant 157
kICMCompressionSessionPropertyID_TimeScale constant 156
kICMDecompressionFrameOptionsPropertyID_-DestinationPixelBuffer constant 157
kICMDecompressionSessionOptionsPropertyID_Accuracy constant 157
kICMDecompressionSessionOptionsPropertyID_-DecompressorComponent constant 157
kICMDecompressionSessionOptionsPropertyID_-DisplayOrderRequired constant 157
kICMDecompressionSessionOptionsPropertyID_-FieldMode constant 157
kICMDecompressionSessionOptionsPropertyID_-MaxBufferCount constant 158
kICMDecompressionSessionOptionsPropertyID_-OutputAheadTime constant 158
kICMDecompressionSessionPropertyID_-NonScheduledDisplayDirection constant 158
kICMDecompressionSessionPropertyID_-NonScheduledDisplayTime constant 158

kICMDecompressionSessionPropertyID_PixelBufferPool constant 158
kICMDecompressionSessionPropertyID_-PixelBufferPoolIsShared constant 158
kICMImageDescriptionPropertyID_ClassicTrackHeight constant 159
kICMImageDescriptionPropertyID_ClassicTrackWidth constant 159
kICMImageDescriptionPropertyID_CleanAperture constant 158
kICMImageDescriptionPropertyID_DisplayHeight constant 159
kICMImageDescriptionPropertyID_DisplayWidth constant 159
kICMImageDescriptionPropertyID_EncodedHeight constant 158
kICMImageDescriptionPropertyID_EncodedWidth constant 158
kICMImageDescriptionPropertyID_FieldInfo constant 159
kICMImageDescriptionPropertyID_GammaLevel constant 159
kICMImageDescriptionPropertyID_NCLCColorInfo constant 159
kICMImageDescriptionPropertyID_PixelAspectRatio constant 158
kICMImageDescriptionPropertyID_-ProductionDisplayHeight constant 159
kICMImageDescriptionPropertyID_-ProductionDisplayWidth constant 159
kICMImageDescriptionPropertyID_RowBytes constant 159
kICMPixelFormatIsIndexed constant 384
kICMPixelFormatIsPlanarMask constant 384
kICMPixelFormatIsSupportedByQD constant 384
kICMSequenceUserPreferredCodecs constant 384
kICMValidTime_DecodeDurationIsValid 161
kICMValidTime_DisplayDurationIsValid constant 162
kICMValidTime_DisplayTimeStampIsValid constant 162
kInstrumentMatchGMNumber 1781
kKnobBasic 1782
kKnobFixedPoint16 constant 1783
kKnobFixedPoint8 constant 1783
kKnobGroupStart constant 1783
kKnobInterruptUnsafe constant 1783
kKnobKeyrangeOverride constant 1783
kKnobReadOnly constant 1782
kKnobTypeBoolean constant 1783
kKnobTypeGroupName constant 1783
kKnobTypeHertz constant 1784
kKnobTypeInstrument constant 1784
kKnobTypeMilliseconds constant 1784
kKnobTypeNote constant 1783
kKnobTypeNumber constant 1783
kKnobTypePan constant 1783
kKnobTypePercentage constant 1784
kKnobTypeSetting constant 1784
kMCIESoundThumb constant 1226
kMediaPacketizerCanPackBalance constant 1958
kMediaPacketizerCanPackEditRate constant 1958
kMediaPacketizerCanPackGraphicsMode constant 1958
kMediaPacketizerCanPackLayer constant 1958
kMediaPacketizerCanPackVolume constant 1958
kMovieMediaBackgroundColor constant 370
kMovieMediaDefaultDataReferenceID constant 370
kMovieMediaPrerollTime constant 371
kMovieMediaSlaveGraphicsMode constant 370
kMovieMediaSlaveTime constant 370
kMusicPacketPortFound constant 1784
kMusicPacketPortLost constant 1784
KnobDescription structure 1767
kPickDontMix 1785
kQTAlphaMode constant 412, 751
kQTAlphaModePreMulColor constant 412, 752
kQTCCIR601VideoGammaLevel constant 752
kQTEnableExif constant 371
kQTParseTextHREFText constant 371
kQTPNGFilterPreference constant 1083
kQTPNGInterlaceStyle constant 1083
kQTPropertyClass_ICMCompressionSession constant 156
kQTPropertyClass_ICMCompressionSessionOptions constant 152
kQTPropertyClass_SampleTable 1278
kQTSAllStatisticsType constant 1959
kQTSDontGetDataStatisticsFlag 1960
kQTSGetUnitsStatisticsFlag constant 1960
kQTSInstantOnFlag_Enable constant 1961
kQTSMemAllocAllocatedInSystemMem constant 1962
kQTSShortStatisticsType constant 1959
kQTSStatisticsFixedDataFormat 1963
kQTSStatisticsFramesPerSecUnitsType constant 1963
kQTSStatisticsOSTypeDataFormat constant 1963
kQTTIFFCompressionMethod constant 1084
kQTTIFFCompression_PackBits constant 1084
kQTTIFFExifGPSUserDataPrefix constant 751
kQTTIFFExifUserDataPrefix constant 751
kQTTIFFLittleEndian constant 1084
kQTTIFFUserDataGeoAsciiParams constant 1085
kQTTIFFUserDataGeoDoubleParams constant 1085
kQTTIFFUserDataGeoKeyDirectory constant 1085
kQTTIFFUserDataIntergraphMatrix constant 1085

kQTTIFFUserDataModelPixelScale constant 1084
kQTTIFFUserDataModelTiepoint constant 1084
kQTTIFFUserDataModelTransformation constant 1084
kQTTIFFUserDataOrientation constant 1085
kQTTIFFUserDataPrefix constant 751
kQTTIFFUserDataPrimaryChromaticities constant 1085
kQTTIFFUserDataTransferFunction constant 1085
kQTTIFFUserDataTransferRange constant 1085
kQTTIFFUserDataWhitePoint constant 1085
kQTTIFFUserDataYCbCrPositioning constant 1085
kQTVRAllModes 2082
kQTVRBackBufferAlwaysRefresh 2075
kQTVRBackBufferAlwaysRefresh constant 2076
kQTVRBackBufferEveryIdle constant 2076
kQTVRBackBufferEveryUpdate constant 2076
kQTVRCube 2083
kQTVRDontLoopViewFrames 2079
kQTVRDown 2080
kQTVRDown constant 2080
kQTVRGetHotSpotTypeSelector 2080
kQTVRGetHotSpotTypeSelector constant 2082
kQTVRImagingCorrection 2078
kQTVRLeft constant 2080
kQTVRMinimumCache constant 2077
kQTVRMouseDownSelector constant 2082
kQTVRMouseEnterSelector constant 2081
kQTVRMouseLeaveSelector constant 2082
kQTVRMouseStillDownSelector constant 2082
kQTVRMouseUpSelector constant 2082
kQTVRMouseWithinSelector constant 2081
kQTVRQuarterRes constant 2077
kQTVRSetFieldOfViewSelector constant 2081
kQTVRSetPanAngleSelector constant 2081
kQTVRSetTiltAngleSelector constant 2081
kQTVRSetViewCenterSelector constant 2081
kQTVRStdCursorType constant 2083
kQTVRSuggestedCache constant 2077
kQTVRTriggerHotSpotSelector constant 2082
kQTVRUseDefaultCursor constant 2083
kRTPInfo_FormatString 1964
kRTPMPHasUserSettingsDialogCharacteristic 1964
kRTPMPRespectDurationFlag 1965
kRTPMPSyncSampleFlag constant 1966
kScreenFloodMethodAlpha constant 964
kScreenFloodMethodKeyColor constant 964
kScreenFloodMethodNone constant 963
kSetAtomicInstCallerGuarantees 1785
kSHExtendedChunkFlag_HasSampleCount constant 1967
kSoftSynthComponentSubType constant 408
kSorenson3CodecType constant 411
kSynthesizerConnectionFMS 1785
kSynthesizerConnectionFMS constant 1786
kSynthesizerConnectionMMgr constant 1786
kSynthesizerConnectionOMS constant 1786
kSynthesizerConnectionQT constant 1786
kSynthesizerDLS 1786
kSynthesizerDynamicChannel constant 1788
kSynthesizerDynamicVoice constant 1787
kSynthesizerGM constant 1788
kSynthesizerHardware constant 1788
kSynthesizerHasSamples constant 1787
kSynthesizerHogsSystemChannel constant 1788
kSynthesizerMicrotone constant 1787
kSynthesizerMixedDrums constant 1787
kSynthesizerOffline constant 1788
kSynthesizerSlowSetPart constant 1788
kSynthesizerSoftware constant 1788
kSynthesizerUsesMIDIPort constant 1787
kTargetChildMovieMovieName constant 371
kTargetChildMovieTrackIndex constant 371
kTargetCurrentQTEventParams constant 372
kTargetQD3DNamedObjectName constant 371
kTargetSpriteID constant 371
kTargetSpriteName constant 371
kTargetTrackIndex constant 371
kTargetTrackType constant 371
kTextEditState constant 406
kTextHyperTextColor constant 406
kTextHyperTextFace constant 406
kTextKeyEntry constant 406
kTextLength constant 406
kTextMouseDown constant 406
kTextRelativeScroll constant 406
kTextScroll constant 406
kTextSelection constant 406
kTextTextBox constant 406
kTuneDontClipNotes 1789

L

langIrishGaelic constant 439
langIrishGaelicScript constant 439
langSimpChinese constant 440
langTradChinese constant 440
LevelMeterInfo structure 1167
LevelMeterInfoPtr data type 1168
linearMatrixType constant 1958
linearTranslateMatrixType constant 1958
LoadMediaIntoRam function 245
LoadMovieIntoRam function 246
LoadTrackIntoRam function 247
Localization Codes 434

loopTimeBase 358

M

MakeFilePreview function 688
MakeImageDescriptionForEffect function 689
MakeImageDescriptionForPixMap function 690
MakeMediaTimeTable function 1380
MakeThumbnailFromPicture function 691
MakeThumbnailFromPictureFile function 692
MakeThumbnailFromPixMap function 693
MakeTrackTimeTable function 1382
MapMatrix function 694
matrixErr constant 427
matrixFlagScale1x constant 964
matrixFlagScale2x constant 964
MCActionFilterProc callback 1223
MCActionFilterUPP data type 1224
MCActionFilterWithRefConProc callback 1223
MCActionFilterWithRefConUPP data type 1224
MCActivate function 1184
MCAddMovieSegment function 1185
MCAdjustCursor function 1186
MCAdjustCursor Values 1227
MCClear function 1187
MCClick function 1187
MCCopy function 1188
MCCut function 1189
MCDoAction function 1190
MCDraw function 1190
MCDrawBadge function 1191
MCEnableEditing function 1192
mcFlagSuppressMovieFrame constant 1226
mcFlagSuppressSpeakerButton constant 1227
mcFlagSuppressStepButtons constant 1226
mcFlagsUseWindowPalette constant 1227
MCGetClip function 1193
MCGetControllerBoundsRect function 1193
MCGetControllerBoundsRgn function 1194
MCGetControllerInfo function 1195
MCGetControllerPort function 1196
MCGetCurrentTime function 1196
MCGetDoActionsProc function 1197
MCGetIndMovie function 1198
MCGetInterfaceElement function 1198
MCGetMenuString function 1199
MCGetMenuString Values 1227
MCGetVisible function 1200
MCGetWindowRgn function 1201
MCIdle function 1202
MCInterfaceElement data type 1225
MCInvalidate function 1202
MCIsControllerAttached function 1203
MCIsEditingEnabled function 1204
MCIsPlayerEvent function 1204
MCKey function 1206
MCMovieChanged function 1206
MCNewAttachedController function 1207
MCPaste function 1208
MCPositionController function 1208
MCPositionController Values 1227
MCPtInController function 1209
MCRemoveAllMovies function 1210
MCRemoveAMovie function 1211
MCRemoveMovie function 1211
MCSetActionFilter function 1212
MCSetActionFilterWithRefCon function 1212
MCSetClip function 1213
MCSetControllerAttached function 1214
MCSetControllerBoundsRect function 1215
MCSetControllerCapabilities function 1215
MCSetControllerPort function 1216
MCSetDuration function 1217
MCSetIdleManager function 1218
MCSetMovie function 1218
MCSetUpEditMenu function 1219
MCSetVisible function 1220
MCTrimMovieSegment function 1220
MCUndo function 1221
Media Characteristics 1523
Media Identifiers 402, 1666
Media Task Flags 1172
Media3DGetCameraAngleAspect function 248
Media3DGetCameraData function 248
Media3DGetCameraRange function 248
Media3DGetCurrentGroup function 249
Media3DGetNamedObjectList function 249
Media3DGetRendererList function 249
Media3DGetViewObject function 250
Media3DRotateNamedObjectTo function 250
Media3DScaleNamedObjectTo function 251
Media3DSetCameraAngleAspect function 251
Media3DSetCameraData function 251
Media3DSetCameraRange function 252
Media3DTranslateNamedObjectTo function 252
MediaChangedNonPrimarySource function 1094
MediaCompare function 1094
MediaContainsDisplayOffsets function 1625
MediaCurrentMediaQueuedData function 1095
MediaDecodeTimeToSampleNum function 1625
MediaDisplayTimeToSampleNum function 1626
MediaDisposeTargetRefCon function 1096
MediaDoIdleActions function 1096
MediaEmptyAllPurgeableChunks function 1097
MediaEmptySampleCache function 1097

MediaEnterEmptyEdit function 1098
MediaEQSpectrumBandsRecord structure 1168
MediaEQSpectrumBandsRecordPtr data type 1168
MediaFlushNonPrimarySourceData function 1098
MediaForceUpdate function 1099
MediaForceUpdate Values 1170
MediaGetActionsForQTEvent function 1099
MediaGetChunkManagementFlags function 1100
MediaGetClock function 1101
MediaGetDrawingRgn function 1101
MediaGetEffectiveSoundBalance function 1102
MediaGetEffectiveVolume function 1102
MediaGetErrorString function 1103
MediaGetGraphicsMode function 1104
MediaGetInvalidRegion function 1104
MediaGetMediaInfo function 1105
MediaGetMediaLoadState function 1105
MediaGetName function 1106
MediaGetNextBoundsChange function 1107
MediaGetNextStepTime function 1108
MediaGetOffscreenBufferSize function 1108
MediaGetPublicInfo function 1109
MediaGetPurgeableChunkMemoryAllowance function 1110
MediaGetSampleDataPointer function 1111
MediaGetSoundBalance function 1111
MediaGetSoundBassAndTreble function 1112
MediaGetSoundEqualizerBandLevels function 1113
MediaGetSoundEqualizerBands function 1113
MediaGetSoundLevelMeterInfo function 1114
MediaGetSoundLevelMeteringEnabled function 1114
MediaGetSoundOutputComponent function 1115
MediaGetSrcRgn function 1115
MediaGetTrackOpaque function 1116
MediaGetURLLink function 1117
MediaGetUserPreferredCodecs function 1118
MediaGetVideoParam function 1118
MediaGGetIdleManager function 1119
MediaGGetStatus function 1120
MediaGSetActiveSegment function 1120
MediaGSetIdleManager function 1121
MediaGSetVolume function 1122
MediaHandlerComponent data type 1661
MediaHasCharacteristic function 1122
MediaHitTestForTargetRefCon function 1123
MediaHitTestTargetRefCon function 1124
MediaHitTestTargetRefCon Values 1173
MediaIdle function 1125
MediaInitialize function 1127
MediaInvalidateRegion function 1128
MediaMakeMediaTimeTable function 1129
MediaMCIsPlayerEvent function 1130
MediaNavigateTargetRefCon function 1131
MediaNavigateTargetRefCon Values 1172
MediaPacketizerRequirements structure 1932
MediaPacketizerRequirements Values 1957
MediaPacketizerRequirementsPtr data type 1933
MediaPrePrerollBegin function 1131
MediaPrePrerollCancel function 1132
MediaPreroll function 1132
MediaPutMediaInfo function 1133
MediaQueueNonPrimarySourceData function 1134
MediaRefConGetProperty function 1135
MediaRefConSetProperty function 1136
MediaRefConSetProperty Values 1172
MediaReleaseSampleDataPointer function 1136
MediaResolveTargetRefCon function 1137
MediaSampleDescriptionB2N function 1138
MediaSampleDescriptionChanged function 1138
MediaSampleDescriptionN2B function 1139
mediaSampleNotSync constant 389, 1281
MediaSetActionsCallback function 1139
MediaSetActive function 1140
MediaSetChunkManagementFlags function 1141
MediaSetChunkManagementFlags Values 1171
MediaSetClip function 1141
MediaSetDimensions function 1142
MediaSetDoMCActionCallback function 1143
MediaSetGraphicsMode function 1143
MediaSetGWorld function 1144
MediaSetHandlerCapabilities function 1145
MediaSetHints function 1146
MediaSetMatrix function 1147
MediaSetMediaTimeScale function 1147
MediaSetMovieTimeScale function 1148
MediaSetNonPrimarySourceData function 1149
MediaSetPublicInfo function 1151
MediaSetPurgeableChunkMemoryAllowance function 1152
MediaSetRate function 1152
MediaSetScreenLock function 1153
MediaSetSoundBalance function 1154
MediaSetSoundBassAndTreble function 1154
MediaSetSoundEqualizerBands function 1155
MediaSetSoundLevelMeteringEnabled function 1156
MediaSetSoundLocalizationData function 1156
MediaSetSoundOutputComponent function 1157
MediaSetTrackInputMapReference function 1157
MediaSetUserPreferredCodecs function 1158
MediaSetVideoParam function 1159
MediaSetVideoParam Values 1171
MediaTargetRefConsEqual function 1160
MediaTimeBaseChanged function 1161
MediaTimeToSampleNum function 1627
MediaTrackEdited function 1161
MediaTrackPropertyAtomChanged function 1162

MediaTrackReferencesChanged function 1162
MediaVideoOutputChanged function 1163
MIDIImportGetSettings function 464
MIDIImportSetSettings function 465
MIDIImportSetSettings Values 565
ModalFilterYDProc callback 1080
ModalFilterYDUPP data type 1081
Movie Controller Options 1225
MovieAudioExtractionBegin function 1383
MovieAudioExtractionEnd function 1384
MovieAudioExtractionFillBuffer function 1384
MovieAudioExtractionGetProperty function 1385
MovieAudioExtractionGetPropertyInfo function 1386
MovieAudioExtractionSetProperty function 1387
MovieEditState data type 1661
MovieEditStateRecord structure 1661
MovieExecuteWiredActions function 1388
MovieExecuteWiredActions Values 1522
MovieExecuteWiredActionsProc callback 1505
MovieExecuteWiredActionsUPP data type 1509
MovieExportAddDataSource function 466
MovieExportComponent data type 555
MovieExportDisposeGetDataAndPropertiesProcs function 467
MovieExportDoUserDialog function 468
movieExportDuration 565
movieExportDuration constant 566
MovieExportFromProceduresToDataRef function 469
MovieExportGetAuxiliaryData function 470
MovieExportGetCreatorType function 470
MovieExportGetDataProc callback 551
MovieExportGetDataUPP data type 555
MovieExportGetFileNameExtension function 471
MovieExportGetPropertyProc callback 552
MovieExportGetPropertyUPP data type 556
MovieExportGetSettingsAsAtomContainer function 471
MovieExportGetShortFileTypeString function 472
MovieExportGetSourceMediaType function 473
movieExportHeight constant 566
MovieExportNewGetDataAndPropertiesProcs function 473
MovieExportSetGetMoviePropertyProc function 475
MovieExportSetProgressProc function 476
MovieExportSetSampleDescription function 476
MovieExportSetSettingsFromAtomContainer function 477
MovieExportToDataRef function 479
MovieExportToFile function 480
MovieExportToHandle function 481
MovieExportValidate function 482
movieExportVideoFilter constant 566
movieExportWidth constant 566
movieFileSpecValid 389, 1666
MovieImportComponent data type 556
MovieImportDataRef function 483
MovieImportDataRef Values 389, 566
MovieImportDoUserDialog function 484
MovieImportDoUserDialogDataRef function 485
MovieImportEstimateCompletionTime function 486
MovieImportFile function 486
MovieImportGetAuxiliaryDataType function 488
MovieImportGetDestinationMediaType function 489
MovieImportGetDontBlock function 489
MovieImportGetFileType function 490
MovieImportGetLoadState function 490
MovieImportGetMaxLoadedTime function 491
MovieImportGetMIMETypeList function 492
MovieImportGetSampleDescription function 492
MovieImportGetSettingsAsAtomContainer function 493
MovieImportHandle function 493
MovieImportIdle function 495
movieImportResultNeedIdles constant 390, 566
MovieImportSetAuxiliaryData function 495
MovieImportSetChunkSize function 496
MovieImportSetDimensions function 497
MovieImportSetDontBlock function 497
MovieImportSetDuration function 498
MovieImportSetFromScrap function 499
MovieImportSetIdleManager function 499
MovieImportSetMediaDataRef function 500
MovieImportSetMediaFile function 501
MovieImportSetNewMovieFlags function 501
MovieImportSetOffsetAndLimit function 502
MovieImportSetOffsetAndLimit64 function 503
MovieImportSetProgressProc function 503
MovieImportSetSampleDescription function 504
MovieImportSetSampleDuration function 505
MovieImportSetSettingsFromAtomContainer function 505
MovieImportValidate function 506
MovieImportValidateDataRef function 507
MovieMediaGetChildDoMCActionCallback function 252
MovieMediaGetChildMovieDataReference function 253
MovieMediaGetCurrentMovieProperty function 254
MovieMediaGetCurrentMovieProperty Values 357
MovieMediaGetCurrentTrackProperty function 255
MovieMediaGetDoMCActionCallback function 255
MovieMediaLoadChildMovieFromDataReference function 256
MovieMediaSetChildMovieDataReference function 257

movieProgressClose constant 391
movieProgressOpen constant 390
MovieProgressProc Values 390
movieProgressUpdatePercent constant 391
MovieRgnCoverProc callback 1506
MovieRgnCoverUPP data type 1510
MovieSearchText function 1389
MovieSearchText Values 1523
MoviesTask function 257
MusicComponent data type 1768
MusicController data type 1769
MusicDerivedCloseResFile function (Deprecated in Mac OS X v10.5) 1678
MusicDerivedMIDISend function (Deprecated in Mac OS X v10.5) 1678
MusicDerivedOpenResFile function (Deprecated in Mac OS X v10.5) 1679
MusicDerivedSetInstrument function (Deprecated in Mac OS X v10.5) 1679
MusicDerivedSetKnob function (Deprecated in Mac OS X v10.5) 1680
MusicDerivedSetMIDI function (Deprecated in Mac OS X v10.5) 1681
MusicDerivedSetPart function (Deprecated in Mac OS X v10.5) 1682
MusicDerivedSetPartInstrumentNumber function (Deprecated in Mac OS X v10.5) 1682
MusicDerivedStorePartInstrument function (Deprecated in Mac OS X v10.5) 1683
MusicFindTone function (Deprecated in Mac OS X v10.5) 1684
MusicGenericConfigure function (Deprecated in Mac OS X v10.5) 1685
MusicGenericGetKnobList function (Deprecated in Mac OS X v10.5) 1686
MusicGenericGetPart function (Deprecated in Mac OS X v10.5) 1687
MusicGenericSetResourceNumbers function (Deprecated in Mac OS X v10.5) 1687
MusicGetDescription function (Deprecated in Mac OS X v10.5) 1688
MusicGetDeviceConnection function (Deprecated in Mac OS X v10.5) 1689
MusicGetDrumKnobDescription function (Deprecated in Mac OS X v10.5) 1689
MusicGetDrumNames function (Deprecated in Mac OS X v10.5) 1690
MusicGetInfoText function (Deprecated in Mac OS X v10.5) 1691
MusicGetInstrumentAboutInfo function (Deprecated in Mac OS X v10.5) 1691
MusicGetInstrumentInfo function (Deprecated in Mac OS X v10.5) 1692
MusicGetInstrumentInfo Values 1781
MusicGetInstrumentKnobDescription function (Deprecated in Mac OS X v10.5) 1693
MusicGetInstrumentNames function (Deprecated in Mac OS X v10.5) 1693
MusicGetKnob function (Deprecated in Mac OS X v10.5) 1694
MusicGetKnobDescription function (Deprecated in Mac OS X v10.5) 1695
MusicGetKnobSettingStrings function (Deprecated in Mac OS X v10.5) 1696
MusicGetMasterTune function (Deprecated in Mac OS X v10.5) 1696
MusicGetMIDIPorts function (Deprecated in Mac OS X v10.5) 1697
MusicGetMIDIProc function (Deprecated in Mac OS X v10.5) 1698
MusicGetPart function (Deprecated in Mac OS X v10.5) 1698
MusicGetPartAtomicInstrument function (Deprecated in Mac OS X v10.5) 1699
MusicGetPartController function (Deprecated in Mac OS X v10.5) 1700
MusicGetPartInstrumentNumber function (Deprecated in Mac OS X v10.5) 1701
MusicGetPartKnob function (Deprecated in Mac OS X v10.5) 1701
MusicGetPartName function (Deprecated in Mac OS X v10.5) 1702
MusicMediaGetIndexedTunePlayer function 1233
MusicMIDIPacket structure 1769
MusicMIDIPacket Values 1784
MusicMIDISendProc callback 1761
MusicMIDISendUPP data type 1769
MusicOfflineDataProc callback 1762
MusicOfflineDataUPP data type 1770
MusicPlayNote function (Deprecated in Mac OS X v10.5) 1702
MusicResetPart function (Deprecated in Mac OS X v10.5) 1703
MusicSendMIDI function (Deprecated in Mac OS X v10.5) 1704
MusicSetKnob function (Deprecated in Mac OS X v10.5) 1705
MusicSetMasterTune function (Deprecated in Mac OS X v10.5) 1705
MusicSetMIDIProc function (Deprecated in Mac OS X v10.5) 1706
MusicSetOfflineTimeTo function (Deprecated in Mac OS X v10.5) 1707
MusicSetPart function (Deprecated in Mac OS X v10.5) 1707

MusicSetPartAtomicInstrument function (Deprecated in Mac OS X v10.5) 1708
MusicSetPartAtomicInstrument Values 1781
MusicSetPartController function (Deprecated in Mac OS X v10.5) 1709
MusicSetPartInstrumentNumber function (Deprecated in Mac OS X v10.5) 1710
MusicSetPartInstrumentNumberInterruptSafe function (Deprecated in Mac OS X v10.5) 1710
MusicSetPartKnob function (Deprecated in Mac OS X v10.5) 1711
MusicSetPartName function (Deprecated in Mac OS X v10.5) 1711
MusicSetPartSoundLocalization function (Deprecated in Mac OS X v10.5) 1712
MusicStartOffline function (Deprecated in Mac OS X v10.5) 1713
MusicStorePartInstrument function (Deprecated in Mac OS X v10.5) 1714
MusicTask function (Deprecated in Mac OS X v10.5) 1714
MusicUseDeviceConnection function (Deprecated in Mac OS X v10.5) 1715

N

NACopyrightDialog function (Deprecated in Mac OS X v10.5) 1716
NADisposeNoteChannel function (Deprecated in Mac OS X v10.5) 1717
NAFindNoteChannelTone function (Deprecated in Mac OS X v10.5) 1717
NAGetController function (Deprecated in Mac OS X v10.5) 1718
NAGetIndNoteChannel function (Deprecated in Mac OS X v10.5) 1719
NAGetKnob function (Deprecated in Mac OS X v10.5) 1719
NAGetMIDIPorts function (Deprecated in Mac OS X v10.5) 1720
NAGetNoteChannelInfo function (Deprecated in Mac OS X v10.5) 1721
NAGetNoteRequest function (Deprecated in Mac OS X v10.5) 1722
NAGetRegisteredMusicDevice function (Deprecated in Mac OS X v10.5) 1722
NANewNoteChannel function (Deprecated in Mac OS X v10.5) 1724
NANewNoteChannelFromAtomicInstrument function (Deprecated in Mac OS X v10.5) 1724
NAPickArrangement function (Deprecated in Mac OS X v10.5) 1725
NAPickEditInstrument function (Deprecated in Mac OS X v10.5) 1726
NAPickInstrument function (Deprecated in Mac OS X v10.5) 1728
NAPlayNote function (Deprecated in Mac OS X v10.5) 1729
NAPrerollNoteChannel function (Deprecated in Mac OS X v10.5) 1730
NARegisterMusicDevice function (Deprecated in Mac OS X v10.5) 1730
NAResetNoteChannel function (Deprecated in Mac OS X v10.5) 1731
NASaveMusicConfiguration function (Deprecated in Mac OS X v10.5) 1732
NASendMIDI function (Deprecated in Mac OS X v10.5) 1732
NASetAtomicInstrument function (Deprecated in Mac OS X v10.5) 1733
NASetController function (Deprecated in Mac OS X v10.5) 1734
NASetInstrumentNumber function (Deprecated in Mac OS X v10.5) 1735
NASetInstrumentNumberInterruptSafe function (Deprecated in Mac OS X v10.5) 1735
NASetKnob function (Deprecated in Mac OS X v10.5) 1736
NASetNoteChannelBalance function (Deprecated in Mac OS X v10.5) 1737
NASetNoteChannelSoundLocalization function (Deprecated in Mac OS X v10.5) 1738
NASetNoteChannelVolume function (Deprecated in Mac OS X v10.5) 1738
NAStuffToneDescription function (Deprecated in Mac OS X v10.5) 1739
NATask function (Deprecated in Mac OS X v10.5) 1740
NAUnregisterMusicDevice function (Deprecated in Mac OS X v10.5) 1740
NAUnrollNoteChannel function (Deprecated in Mac OS X v10.5) 1741
New Movie Properties 392
NewActionsUPP function 1390
NewCallBack function 258
NewCDataHandlerUPP function 824
NewCharDataHandlerUPP function 825
NewCommentHandlerUPP function 825
NewDataHCompletionUPP function 825
NewDoMCActionUPP function 1391
NewEndDocumentHandlerUPP function 826
NewEndElementHandlerUPP function 827
NewGetMovieUPP function 1391
NewICMAlignmentUPP function 106
NewICMCompletionUPP function 107
NewICMConvertDataFormatUPP function 107
NewICMCursorShieldedUPP function 108
NewICMDataUPP function 108
NewICMFlushUPP function 109

NewICMMemoryDisposedUPP function 109
NewICMProgressUPP function 110
NewImageCodecDrawBandCompleteUPP function 927
NewImageCodecMPDrawBandUPP function 927
NewImageCodecTimeTriggerUPP function 928
NewImageGWorld function 694
NewMCActionFilterUPP function 1222
NewMCActionFilterWithRefConUPP function 1222
NewMovie function 259
NewMovieController function 1392
NewMovieController Values 392
NewMovieDrawingCompleteUPP function 1393
NewMovieEditState function 1628
NewMovieExecuteWiredActionsUPP function 1393
NewMovieExportGetDataUPP function 508
NewMovieExportGetPropertyUPP function 509
NewMovieExportStageReachedCallbackUPP function 509
NewMovieForDataRefFromHandle function 1394
NewMovieFromDataFork function 1395
NewMovieFromDataFork64 function 1396
NewMovieFromDataRef function 1397
NewMovieFromFile function 1398
NewMovieFromFile Values 1522
NewMovieFromHandle function 1400
NewMovieFromProperties function 260
NewMovieFromScrap function 1402
NewMovieFromStorageOffset function 1402
NewMovieFromUserProc function 1404
NewMoviePrePrerollCompleteUPP function 1405
NewMoviePreviewCallOutUPP function 1406
NewMovieProgressUPP function 1406
NewMovieRgnCoverUPP function 1407
NewMoviesErrorUPP function 1408
NewMovieTrack function 1628
NewMusicMIDISendUPP function (Deprecated in Mac OS X v10.5) 1741
NewMusicOfflineDataUPP function (Deprecated in Mac OS X v10.5) 1742
NewPrePrerollCompleteUPP function 1163
NewPreprocessInstructionHandlerUPP function 827
NewQDPixUPP function 110
NewQTBandwidthNotificationUPP function (Deprecated in Mac OS X v10.4) 1234
NewQTCallBackUPP function 1408
NewQTEffectListFilterUPP function 1409
NewQTNextTaskNeededSoonerCallbackUPP function 1409
NewQTSModalFilterUPP function 1794
NewQTSNotificationUPP function 1794
NewQTSPanelFilterUPP function 1795
NewQTSyncTaskUPP function 1410
NewQTTrackPropertyListenerUPP function 1235
NewQTVRBackBufferImagingUPP function 1978
NewQTVREnteringNodeUPP function 1979
NewQTVRImagingCompleteUPP function 1980
NewQTVRInterceptUPP function 1980
NewQTVRLeavingNodeUPP function 1981
NewQTVRMouseOverHotSpotUPP function 1981
NewRTPMPDataReleaseUPP function 1795
NewRTPPBCallbackUPP function 1795
NewSCModalFilterUPP function 510
NewSCModalHookUPP function 510
NewSGAddFrameBottleUPP function 2101
NewSGCompressBottleUPP function 2101
NewSGCompressCompleteBottleUPP function 2102
NewSGDataUPP function 2102
NewSGDisplayBottleUPP function 2103
NewSGDisplayCompressBottleUPP function 2104
NewSGGrabBottleUPP function 2104
NewSGGrabCompleteBottleUPP function 2105
NewSGGrabCompressCompleteBottleUPP function 2105
NewSGModalFilterUPP function 2106
NewSGTransferFrameBottleUPP function 2106
NewSprite function 1410
NewSpriteWorld function 1411
NewStartDocumentHandlerUPP function 827
NewStartElementHandlerUPP function 828
NewStdPixUPP function 111
NewTextMediaUPP function 1413
NewTimeBase function 261
NewTrackEditState function 1630
NewTrackMedia function 1630
NewTrackTransferUPP function 1414
NewTuneCallBackUPP function (Deprecated in Mac OS X v10.5) 1742
NewTunePlayCallBackUPP function (Deprecated in Mac OS X v10.5) 1743
NewTweenerDataUPP function 1414
NewUserData function 1415
NewUserDataFromHandle function 1415
NewVdigIntUPP function 828
noDMAErr constant 428
noMoreKeyColorsErr constant 428
noRecordOfApp constant 427
NoteAllocator data type 1770
NoteChannel data type 1770
NoteRequest structure 1770
notExactMatrixErr constant 427
notExactSizeErr constant 428
notPatCopy constant 432
notPatOr constant 432
notPatXor constant 432
notSrcBic constant 431
notSrcCopy constant 431

notSrcOr constant 431
notSrcXor constant 431

O

OpenADataHandler function 1631
OpenCPicParams structure 745
OpenMovieFile function 1416
OpenMovieStorage function 1417
OptionBits data type 1225

P

PasteHandleIntoMovie function 1633
PasteMovieSelection function 1634
patBic constant 432
patCopy constant 431
patOr constant 432
patXor constant 432
pdActionCompactSample constant 396
pdActionConvertSettingsToText constant 398
pdActionConvertSettingsToXML constant 398
pdActionConvertSettingsToXMLWithComments constant 398
pdActionConvertXMLToSettings constant 398
pdActionCustomDisposeControl constant 396
pdActionCustomDoEditCommand constant 397
pdActionCustomGetEnableValue constant 397
pdActionCustomGetValue constant 397
pdActionCustomHandleEvent constant 397
pdActionCustomNewControl constant 396
pdActionCustomPositionControl constant 396
pdActionCustomSetEditCallout constant 397
pdActionCustomSetEditMenu constant 397
pdActionCustomSetFocus constant 397
pdActionCustomSetPreviewPicture constant 397
pdActionCustomSetSampleTime constant 397
pdActionCustomShowHideControl constant 396
pdActionDoEditCommand constant 396
pdActionFetchPreview constant 395
pdActionGetDialogSettings constant 395
pdActionGetNextSample constant 395
pdActionGetPreviousSample constant 396
pdActionGetSubPanelMenuValue constant 396
pdActionModelessCallback constant 395
pdActionRunInEventLoop constant 397
pdActionSetDialogSettings constant 395
pdActionSetEditCallout constant 396
pdActionSetPropertyComponent constant 398
pdActionSetSampleTime constant 396
pdOptionsEditCurrentEffectOnly constant 398
pdOptionsHidePreview constant 398
pdOptionsModalDialogBox constant 398
perspectiveMatrixType constant 1958
PlayMoviePreview function 262
PNG Properties 1082
pnotComponent data type 556
PrePrerollCompleteProc callback 1164
PrePrerollCompleteUPP data type 1169
PrePrerollMovie function 263
PrerollMovie function 264
PreviewEvent function 511
PreviewMakePreview function 511
PreviewMakePreviewReference function 512
PreviewShowData function 513
progressOpAddMovieSelection constant 391
progressOpCopy constant 391
progressOpCut constant 392
progressOpFlatten constant 391
progressOpImportMovie constant 392
progressOpInsertMovieSegment constant 391
progressOpInsertTrackSegment constant 391
progressOpLoadMediaIntoRam constant 392
progressOpLoadMovieIntoRam constant 392
progressOpLoadTrackIntoRam constant 392
progressOpPaste constant 391
PtInDSequenceData function 695
PtInMovie function 1634
PtInTrack function 1635
PutMovieForDataRefIntoHandle function 265
PutMovieIntoDataFork function 266
PutMovieIntoDataFork64 function 267
PutMovieIntoHandle function 268
PutMovieIntoStorage function 268
PutMovieIntoTypedHandle function 1636
PutMovieOnScrap function 1418
PutMovieOnScrap Values 1522
PutUserDataIntoHandle function 1419

Q

QDPixProc callback 134
QDPixUPP data type 139
QHdr structure 746
QHdrPtr data type 747
QTAddComponentPropertyListener function 111
QTAddMovieError function 1420
QTAddMoviePropertyListener function 1235
QTAddTrackPropertyListener function 1236
QTAtomSpec structure 1934
QTAtomSpecPtr data type 1934
QTAtomType data type 1510

QTAudioContextCreateForAudioDevice function 269
QTAudioFrequencyLevels structure 1510
QTAudioVolumeLevels structure 1511
QTBandwidthNotificationProc callback 1276
QTBandwidthNotificationUPP data type 1276
QTBandwidthReference data type 1277
QTBandwidthRelease function (Deprecated in Mac OS X v10.4) 1236
QTBandwidthRequest function (Deprecated in Mac OS X v10.4) 1237
QTBandwidthRequestForTimeBase function (Deprecated in Mac OS X v10.4) 1238
QTComponentPropertyListenerCollectionAddListener function 113
QTComponentPropertyListenerCollectionContext structure 139
QTComponentPropertyListenerCollectionCreate function 114
QTComponentPropertyListenerCollectionHasListeners-ForProperty function 114
QTComponentPropertyListenerCollectionIsEmpty function 115
QTComponentPropertyListenerCollectionNotify-Listeners function 116
QTComponentPropertyListenerCollectionRemove-Listener function 117
QTCopyAtom function 1421
QTCopyAtomDataToHandle function 1421
QTCopyAtomDataToPtr function 1422
QTCopyMediaMetaData function 1239
QTCopyMovieMetaData function 1239
QTCopyTrackMetaData function 1240
QTCountChildrenOfType function 1424
QTCreateStandardParameterDialog function 1424
QTCreateUUID function 1426
QTCustomActionTargetPtr data type 1169
QTCustomActionTargetRecord structure 1169
QTDismissStandardParameterDialog function 1426
QTDisposeAtomContainer function 1427
QTDisposeTween function 1428
QTDoTween function 1428
QTDoTweenPtr function 1429
QTEffectListFilterProc callback 1506
QTEffectListFilterUPP data type 1511
QTEffectListOptions data type 1511
QTEqualUUIDs function 1430
QTErrorReplacementPtr data type 1511
QTErrorReplacementRecord structure 1512
QTFindChildByID function 1430
QTFindChildByIndex function 1431
QTFloatSingle data type 352
QTGetAccessKeys function 1432
QTGetAtomDataPtr function 1433
QTGetAtomParent function 1434
QTGetAtomTypeAndID function 1435
QTGetComponentProperty function 118
QTGetComponentPropertyInfo function 120
QTGetDataHandlerDirectoryDataReference function 1436
QTGetDataHandlerFullPathCFString function 1436
QTGetDataHandlerTargetNameCFString function 1437
QTGetDataReferenceDirectoryDataReference function 1438
QTGetDataReferenceFullPathCFString function 1438
QTGetDataReferenceTargetNameCFString function 1439
QTGetDataRefMaxFileOffset function 1440
QTGetEffectsList function 1441
QTGetEffectsList Values 1517
QTGetEffectsListExtended function 1442
QTGetEffectSpeed function 1443
QTGetEffectSpeed Values 1517
QTGetFileNameExtension function 696
QTGetMIMETypeInfo function 1637
QTGetMIMETypeInfo Values 1665
QTGetMovieProperty function 1241
QTGetMoviePropertyInfo function 1241
QTGetMovieRestrictions function 1444
QTGetNextChildType function 1445
QTGetPixelFormatDepthForImageDescription function 696
QTGetPixelSize function 697
QTGetPixMapHandleGammaLevel function 698
QTGetPixMapHandleRequestedGammaLevel function 698
QTGetPixMapHandleRowBytes function 699
QTGetPixMapPtrGammaLevel function 699
QTGetPixMapPtrRequestedGammaLevel function 700
QTGetPixMapPtrRowBytes function 700
QTGetSupportedRestrictions function 1445
QTGetTimeUntilNextTask function 270
QTGetTrackProperty function 1242
QTGetTrackPropertyInfo function 1243
QTGetWallClockTimeBase function 271
QTIdleManagerClose function 271
QTIdleManagerGetNextIdleTime function 272
QTIdleManagerNeedsAnIdle function 272
QTIdleManagerOpen function 273
QTIdleManagerSetNextIdleTime function 273
QTIdleManagerSetNextIdleTimeDelta function 274
QTIdleManagerSetNextIdleTimeNever function 275
QTIdleManagerSetNextIdleTimeNow function 275
QTIdleManagerSetParent function 276
QTInsertChild function 1446

QTInsertChildren function 1447
QTInstallNextTaskNeededSoonerCallback function 277
QTIsStandardParameterDialogEvent function 1448
QTLockContainer function 1449
QTMetaDataAddItem function 1244
QTMetaDataGetItemProperty function 1246
QTMetaDataGetItemPropertyInfo function 1247
QTMetaDataGetItemValue function 1248
QTMetaDataGetNextItem function 1248
QTMetaDataGetProperty function 1250
QTMetaDataGetPropertyInfo function 1250
QTMetaDataRelease function 1251
QTMetaDataRemoveItem function 1252
QTMetaDataRemoveItemsWithKey function 1252
QTMetaDataRetain function 1253
QTMetaDataSetItem function 1254
QTMetaDataSetItemProperty function 1254
QTMetaDataSetProperty function 1255
QTMIDIComponent data type 1771
QTMIDIGetMIDIPorts function (Deprecated in Mac OS X v10.5) 1743
QTMIDIPortListHandle data type 1771
QTMIDIPortListPtr data type 1771
QTMIDISendMIDI function (Deprecated in Mac OS X v10.5) 1744
QTMIDIUseSendPort function (Deprecated in Mac OS X v10.5) 1745
QTMLCreateMutex function 2361
QTMLDestroyMutex function 2362
qtmlDllEntryNotFoundErr constant 427
qtmlDllLoadErr constant 427
QTMLGrabMutex function 2362
QTMLMutex data type 2365
QTMLReturnMutex function 2363
QTMLTryGrabMutex function 2363
QTMLYieldCPU function 2364
QTMLYieldCPUTime function 2364
QTMLYieldCPUTime Values 2365
QTMovieNeedsTimeTable function 1450
QTNewAlias function 1451
QTNewAtomContainer function 1451
QTNewDataReferenceFromCFURL function 1452
QTNewDataReferenceFromFSRef function 1453
QTNewDataReferenceFromFSRefCFString function 1454
QTNewDataReferenceFromFSSpec function 1455
QTNewDataReferenceFromFullPathCFString function 1456
QTNewDataReferenceFromFullPathCFString Values 1521
QTNewDataReferenceFromURLCFString function 1457
QTNewDataReferenceWithDirectoryCFString function 1458
QTNewGWorld function 701
QTNewGWorldFromPtr function 703
QTNewMoviePropertyElement structure 352
QTNewTween function 1459
QTNextChildAnyType function 1460
QTOpenGLTextureContextCreate function 121
qtParamErr constant 427
QTParameterValidationOptions data type 956
QTParseTextHREF function 277
QTPhotoDefineHuffmanTable function 928
QTPhotoDefineQuantizationTable function 929
QTPhotoSetRestartInterval function 930
QTPhotoSetSampling function 931
QTPixelBufferContextCreate function 122
QTRegisterAccessKey function 1461
QTRemoveAtom function 1461
QTRemoveChildren function 1462
QTRemoveComponentPropertyListener function 122
QTRemoveMoviePropertyListener function 1256
QTRemoveTrackPropertyListener function 1257
QTReplaceAtom function 1463
QTRestrictionSet data type 1512
QTRestrictionSetRecord structure 1512
QTRestrictionsGetIndClass function 1463
QTRestrictionsGetInfo function 1464
QTRestrictionsGetItem function 1465
QTRuntimeSpriteDescPtr data type 353
QTRuntimeSpriteDescStruct structure 353
QTSAllocBuffer function 1796
QTSAllocMemPtr function 1796
QTSampleTableAddSampleDescription function 1257
QTSampleTableAddSampleReferences function 1258
QTSampleTableCopySampleDescription function 1259
QTSampleTableCreateMutable function 1260
QTSampleTableCreateMutableCopy function 1261
QTSampleTableGetDataOffset function 1261
QTSampleTableGetDataSizePerSample function 1262
QTSampleTableGetDecodeDuration function 1262
QTSampleTableGetDisplayOffset function 1263
QTSampleTableGetNextAttributeChange function 1263
QTSampleTableGetNextAttributeChange Values 1279
QTSampleTableGetNumberOfSamples function 1265
QTSampleTableGetProperty function 1265
QTSampleTableGetPropertyInfo function 1266
QTSampleTableGetSampleDescriptionID function 1268
QTSampleTableGetSampleFlags function 1268
QTSampleTableGetSampleFlags Values 388, 1280
QTSampleTableGetTimeScale function 1269
QTSampleTableGetTypeID function 1269
QTSampleTableRelease function 1269

QTSampleTableReplaceRange function 1270
QTSampleTableRetain function 1271
QTSampleTableSetProperty function 1271
QTSampleTableSetTimeScale function 1272
QTScheduledBandwidthPtr data type 1277
QTScheduledBandwidthRecord structure 1277
QTScheduledBandwidthReference data type 1278
QTScheduledBandwidthRelease function (Deprecated in Mac OS X v10.4) 1273
QTScheduledBandwidthRequest function (Deprecated in Mac OS X v10.4) 1273
QTSCopyMessage function 1797
QTSDisposePresentation function 1797
QTSDisposeStatHelper function 1798
QTSDisposeStream function 1798
QTSDuplicateMessage function 1799
QTSDupMessage function 1800
QTSetAtomData function 1465
QTSetAtomID function 1467
QTSetComponentProperty function 123
QTSetComponentProperty Values 384
QTSetMovieProperty function 1274
QTSetPixMapHandleGammaLevel function 704
QTSetPixMapHandleRequestedGammaLevel function 705
QTSetPixMapHandleRowBytes function 706
QTSetPixMapPtrGammaLevel function 706
QTSetPixMapPtrRequestedGammaLevel function 707
QTSetPixMapPtrRequestedGammaLevel Values 752
QTSetPixMapPtrRowBytes function 708
QTSetTrackProperty function 1275
QTSExportParams structure 1934
QTSFindMediaPacketizer function 1800
QTSFindMediaPacketizerForPayloadID function 1801
QTSFindMediaPacketizerForPayloadName function 1801
QTSFindMediaPacketizerForTrack function 1802
QTSFindReassemblerForPayloadID function 1803
QTSFindReassemblerForPayloadName function 1803
QTSFlattenMessage function 1804
QTSFreeMessage function 1805
QTSGetErrorString function 1805
QTSGetNetworkAppName function 1806
QTSGetOrMakeStatAtomForStream function 1806
QTSGetStreamPresentation function 1807
QTSInitializeMediaParams function 1808
QTSInsertStatistic function 1808
QTSInsertStatisticName function 1809
QTSInsertStatisticUnits function 1810
QTSInsertStatisticUnits Values 1962
QTSInstantOnPref structure 1935
QTSInstantOnPref Values 1961
QTSMediaGetIndStreamInfo function 1811
QTSMediaGetInfo function 1812
QTSMediaParams structure 1936
QTSMediaSetIndStreamInfo function 1813
QTSMediaSetInfo function 1814
QTSMediaSetInfo Values 1961
QTSMemPtr data type 1936
QTSMessageLength function 1814
QTSNewHandle function 1815
QTSNewPresentation function 1816
QTSNewPresentationFromData function 1816
QTSNewPresentationFromDataRef function 1817
QTSNewPresentationFromFile function 1817
QTSNewPresentationParams structure 1937
QTSNewPtr function 1818
QTSNewPtr Values 1961
QTSNewSourcer function 1819
QTSNewStatHelper function 1820
QTSNewStreamBuffer function 1821
QTSNoProxyPref structure 1938
QTSNotificationProc callback 1931
QTSNotificationUPP data type 1938
QTSoundDescriptionConvert function 278
QTSoundDescriptionCreate function 279
QTSoundDescriptionGetProperty function 280
QTSoundDescriptionGetPropertyInfo function 281
QTSoundDescriptionSetProperty function 282
QTSPrefsAddConnectionSetting function 1821
QTSPrefsAddProxySetting function 1822
QTSPrefsAddProxyUserInfo function 1823
QTSPrefsFindConnectionByType function 1823
QTSPrefsFindProxyByType function 1824
QTSPrefsFindProxyUserInfoByType function 1825
QTSPrefsGetActiveConnection function 1826
QTSPrefsGetActiveConnection Values 1960
QTSPrefsGetInstantOnSettings function 1827
QTSPrefsGetNoProxyURLs function 1827
QTSPrefsSetInstantOnSettings function 1827
QTSPrefsSetNoProxyURLs function 1828
QTSPresAddSourcer function 1829
QTSPresentation data type 1939
QTSPresentationRecord structure 1939
QTSPresExport function 1829
QTSPresGetActiveSegment function 1830
QTSPresGetClip function 1830
QTSPresGetDimensions function 1831
QTSPresGetEnable function 1832
QTSPresGetFlags function 1832
QTSPresGetFlags Values 1959
QTSPresGetGraphicsMode function 1833
QTSPresGetGWorld function 1834
QTSPresGetIndSourcer function 1834
QTSPresGetIndStream function 1835

QTSPresGetInfo function 1836
QTSPresGetMatrix function 1838
QTSPresGetNotificationProc function 1838
QTSPresGetNumSourcers function 1839
QTSPresGetNumStreams function 1839
QTSPresGetPicture function 1840
QTSPresGetPlayHints function 1840
QTSPresGetPreferredRate function 1841
QTSPresGetPresenting function 1842
QTSPresGetSettings function 1842
QTSPresGetSettingsAsText function 1843
QTSPresGetTimeBase function 1844
QTSPresGetTimeScale function 1844
QTSPresGetVolumes function 1845
QTSPresHasCharacteristic function 1845
QTSPresIdle function 1846
QTSPresIdleParams structure 1939
QTSPresInvalidateRegion function 1847
QTSPresNewStream function 1847
QTSPresParams structure 1940
QTSPresPreroll function 1848
QTSPresPreroll64 function 1848
QTSPresPreview function 1849
QTSPresRemoveSourcer function 1850
QTSPresSetActiveSegment function 1850
QTSPresSetClip function 1851
QTSPresSetDimensions function 1852
QTSPresSetEnable function 1852
QTSPresSetFlags function 1853
QTSPresSetGraphicsMode function 1853
QTSPresSetGWorld function 1854
QTSPresSetInfo function 1855
QTSPresSetInfo Values 1960
QTSPresSetMatrix function 1856
QTSPresSetNotificationProc function 1856
QTSPresSetPlayHints function 1857
QTSPresSetPreferredRate function 1858
QTSPresSetPresenting function 1858
QTSPresSetSettings function 1859
QTSPresSettingsDialog function 1860
QTSPresSettingsDialogWithFilters function 1860
QTSPresSetVolumes function 1861
QTSPresSkipTo function 1862
QTSPresSkipTo64 function 1862
QTSPresStart function 1863
QTSPresStop function 1864
QTSProxyPref structure 1941
QTSReleaseMemPtr function 1864
QTSSetNetworkAppName function 1865
QTSSetNetworkAppName Values 1962
QTSSourcer data type 1941
QTSSourcerGetEnable function 1865
QTSSourcerGetInfo function 1866
QTSSourcerGetTimeScale function 1866
QTSSourcerIdle function 1867
QTSSourcerInitialize function 1867
QTSSourcerInitParams structure 1942
QTSSourcerSetEnable function 1868
QTSSourcerSetInfo function 1868
QTSSourcerSetTimeScale function 1869
QTSStatHelper data type 1942
QTSStatHelperGetNumStats function 1870
QTSStatHelperGetStats function 1870
QTSStatHelperNext function 1871
QTSStatHelperNextParams structure 1943
QTSStatHelperNextParams Values 1962
QTSStatHelperRecord structure 1944
QTSStatHelperResetIter function 1871
QTSStatisticsParams Values 1959
QTSStream data type 1944
QTSStreamBuffer structure 1944
QTSStreamBufferDataInfo function 1872
QTSStreamRecord structure 1946
QTStandardParameterDialogDoAction function 1467
QTSTransportPref structure 1946
QTSTransportPref Values 1959
QTSwapAtoms function 1469
QTSyncTaskProc callback 1507
QTSyncTaskUPP data type 1513
QTTextToNativeText function 282
QTTextToNativeText Values 357
QTTweener data type 1513
QTTweenerRecord structure 1513
QTUninstallNextTaskNeededSoonerCallback function 283
QTUnlockContainer function 1469
QTUnregisterAccessKey function 1470
QTUnregisterAccessKey Values 1519
QTUpdateGWorld function 708
QTUUID structure 1514
QTVideoOutputBaseSetEchoPort function 2258
QTVideoOutputBegin function 2259
QTVideoOutputComponent data type 2347
QTVideoOutputCopyIndAudioOutputDeviceUID function 2260
QTVideoOutputCustomConfigureDisplay function 2260
QTVideoOutputEnd function 2261
QTVideoOutputGetClientName function 2262
QTVideoOutputGetClock function 2262
QTVideoOutputGetCurrentClientName function 2263
QTVideoOutputGetDisplayMode function 2264
QTVideoOutputGetDisplayModeList function 2264
QTVideoOutputGetGWorld function 2265
QTVideoOutputGetGWorldParameters function 2266

QTVideoOutputGetIndImageDecompressor function 2267
QTVideoOutputGetIndSoundOutput function 2267
QTVideoOutputRestoreState function 2268
QTVideoOutputSaveState function 2269
QTVideoOutputSetClientName function 2270
QTVideoOutputSetDisplayMode function 2270
QTVideoOutputSetEchoPort function 2271
QTVisualContextCopyImageForTime function 125
QTVisualContextGetAttribute function 125
QTVisualContextGetTypeID function 126
QTVisualContextIsNewImageAvailable function 126
QTVisualContextRelease function 127
QTVisualContextRetain function 128
QTVisualContextSetAttribute function 128
QTVisualContextSetImageAvailableCallback function 129
QTVisualContextTask function 129
QTVRAnglesToCoord function 1982
QTVRAngularUnits data type 2069
QTVRAreaOfInterest structure 2069
QTVRBackBufferImagingProc callback 2065
QTVRBackBufferImagingUPP data type 2070
QTVRBeginUpdateStream function 1983
QTVRCallInterceptedProc function 1984
QTVRColumnToPan function 1985
QTVRControlSetting data type 2070
QTVRCoordToAngles function 1985
QTVRCursorRecord structure 2070
QTVRCursorRecord Values 2083
QTVREnableFrameAnimation function 1986
QTVREnableHotSpot function 1987
QTVREnableHotSpot Values 2078
QTVREnableTransition function 1988
QTVREnableViewAnimation function 1989
QTVREndUpdateStream function 1989
QTVREnteringNodeProc callback 2065
QTVREnteringNodeUPP data type 2071
QTVRFloatPoint structure 2071
QTVRGetAngularUnits function 1990
QTVRGetAnimationSetting function 1991
QTVRGetAvailableResolutions function 1992
QTVRGetBackBufferMemInfo function 1992
QTVRGetBackBufferSettings function 1994
QTVRGetConstraints function 1996
QTVRGetConstraintStatus function 1996
QTVRGetControlSetting function 1997
QTVRGetCurrentMouseMode function 1998
QTVRGetCurrentNodeID function 1999
QTVRGetCurrentViewDuration function 1999
QTVRGetFieldOfView function 2000
QTVRGetFrameAnimation function 2001
QTVRGetFrameRate function 2001
QTVRGetHotSpotRegion function 2002
QTVRGetHotSpotType function 2003
QTVRGetImagingProperty function 2004
QTVRGetInteractionProperty function 2005
QTVRGetMouseDownTracking function 2006
QTVRGetMouseOverTracking function 2006
QTVRGetNodeInfo function 2007
QTVRGetNodeType function 2008
QTVRGetPanAngle function 2008
QTVRGetQTVRInstance function 2009
QTVRGetQTVRTrack function 2010
QTVRGetTiltAngle function 2011
QTVRGetViewAnimation function 2012
QTVRGetViewCenter function 2013
QTVRGetViewCurrentTime function 2013
QTVRGetViewingLimits function 2014
QTVRGetViewParameter function 2015
QTVRGetViewRate function 2016
QTVRGetViewState function 2016
QTVRGetViewStateCount function 2017
QTVRGetVisible function 2018
QTVRGetVisibleHotSpots function 2018
QTVRGetVRWorld function 2019
QTVRGoToNodeID function 2020
QTVRGoToNodeID Values 2076
QTVRImagingCompleteProc callback 2066
QTVRImagingCompleteUPP data type 2072
QTVRImagingMode data type 2072
QTVRInstallInterceptProc function 2021
QTVRInstance data type 2072
QTVRInteractionNudge function 2022
QTVRInterceptProc callback 2066
QTVRInterceptRecord structure 2072
QTVRInterceptUPP data type 2074
QTVRLeavingNodeProc callback 2067
QTVRLeavingNodeUPP data type 2074
QTVRMouseDown function 2023
QTVRMouseEnter function 2025
QTVRMouseLeave function 2025
QTVRMouseOverHotSpotProc callback 2068
QTVRMouseOverHotSpotUPP data type 2074
QTVRMouseStillDown function 2026
QTVRMouseStillDownExtended function 2027
QTVRMouseUp function 2028
QTVRMouseUpExtended function 2029
QTVRMouseWithin function 2031
QTVRNudge function 2031
QTVRNudgeControl data type 2074
QTVRObjectAnimationSetting data type 2075
QTVRPanToColumn function 2032
QTVRProcSelector data type 2075
QTVRPtToAngles function 2033
QTVRPtToHotSpotID function 2034

QTVRRefreshBackBuffer function 2035
QTVRReplaceCursor function 2035
QTVRRowToTilt function 2036
QTVRSetAngularUnits function 2037
QTVRSetAngularUnits Values 2077
QTVRSetAnimationSetting function 2038
QTVRSetBackBufferImagingProc function 2039
QTVRSetBackBufferPrefs function 2040
QTVRSetBackBufferPrefs Values 2077
QTVRSetConstraints function 2041
QTVRSetControlSetting function 2042
QTVRSetControlSetting Values 2083
QTVRSetEnteringNodeProc function 2043
QTVRSetFieldOfView function 2044
QTVRSetFrameRate function 2045
QTVRSetImagingProperty function 2046
QTVRSetInteractionProperty function 2047
QTVRSetInteractionProperty Values 2078
QTVRSetLeavingNodeProc function 2048
QTVRSetMouseDownTracking function 2049
QTVRSetMouseOverHotSpotProc function 2050
QTVRSetMouseOverTracking function 2051
QTVRSetPanAngle function 2051
QTVRSetPrescreenImagingCompleteProc function 2052
QTVRSetPrescreenImagingCompleteProc Values 2080
QTVRSetTiltAngle function 2054
QTVRSetTransitionProperty function 2055
QTVRSetTransitionProperty Values 2082
QTVRSetViewCenter function 2056
QTVRSetViewCurrentTime function 2056
QTVRSetViewParameter function 2057
QTVRSetViewRate function 2058
QTVRSetViewState function 2059
QTVRSetViewState Values 2076
QTVRSetVisible function 2059
QTVRShowDefaultView function 2060
QTVRTiltToRow function 2061
QTVRTriggerHotSpot function 2061
QTVRUpdate function 2063
QTVRViewStateType data type 2075
QTVRWrapAndConstrain function 2063
QTVRWrapAndConstrain Values 386, 2079
QuadToQuadMatrix function 710
QuickTime Preferences Dialog Options 393

R

RectMatrix function 710
RectPtr data type 2347
RegionCode data type 354
RemoveCallBackFromTimeBase function 284
RemoveImageDescriptionExtension function 711
RemoveMovieExecuteWiredActionsProc function 1471
RemoveMovieResource function 1471
RemoveSoundDescriptionExtension function 1472
RemoveUserData function 1472
RemoveUserDataText function 1473
ReplaceDSequenceImageDescription function 712
RotateMatrix function 712
RTPMediaPacketizer data type 1947
RTPMPDataReleaseProc callback 1931
RTPMPDataReleaseUPP data type 1947
RTPMPDoUserDialog function 1872
RTPMPFlush function 1873
RTPMPGetInfo function 1874
RTPMPGetMaxPacketDuration function 1875
RTPMPGetMaxPacketSize function 1875
RTPMPGetMediaType function 1876
RTPMPGetPacketBuilder function 1876
RTPMPGetSettings function 1877
RTPMPGetSettingsAsText function 1878
RTPMPGetSettingsIntoAtomContainerAtAtom function 1878
RTPMPGetTimeBase function 1879
RTPMPGetTimeScale function 1879
RTPMPHasCharacteristic function 1880
RTPMPIdle function 1881
RTPMPIdle Values 1965
RTPMPInitialize function 1881
RTPMPInitialize Values 1965
RTPMPPreflightMedia function 1882
RTPMPReset function 1883
RTPMPSampleDataParams structure 1947
RTPMPSetInfo function 1884
RTPMPSetMaxPacketDuration function 1885
RTPMPSetMaxPacketSize function 1885
RTPMPSetMediaType function 1886
RTPMPSetPacketBuilder function 1887
RTPMPSetSampleData function 1887
RTPMPSetSettings function 1888
RTPMPSetSettingsFromAtomContainerAtAtom function 1889
RTPMPSetTimeBase function 1890
RTPMPSetTimeScale function 1890
RTPPacketBuilder data type 1949
RTPPacketGroupRef data type 1949
RTPPacketRef data type 1949
RTPPacketRepeatedDataRef data type 1949
RTPPayloadSortRequest structure 1949
RTPPayloadSortRequestPtr data type 1950
RTPPBAddPacketLiteralData function 1891
RTPPBAddPacketRepeatedData function 1892
RTPPBAddPacketSampleData function 1893

RTPPBAddPacketSampleData64 function 1894
RTPPBAddRepeatPacket function 1895
RTPPBBeginPacket function 1896
RTPPBBeginPacketGroup function 1897
RTPPBCallbackProc callback 1932
RTPPBCallbackUPP data type 1950
RTPPBEndPacket function 1898
RTPPBEndPacketGroup function 1899
RTPPBGetCallback function 1900
RTPPBGetInfo function 1901
RTPPBGetPacketSequenceNumber function 1901
RTPPBGetPacketTimeStampOffset function 1902
RTPPBGetSampleData function 1903
RTPPBReleaseRepeatedData function 1904
RTPPBSetCallback function 1904
RTPPBSetInfo function 1905
RTPPBSetPacketSequenceNumber function 1906
RTPPBSetPacketTimeStampOffset function 1906
RTPReassembler data type 1950
RTPRssmAdjustPacketParams function 1907
RTPRssmClearCachedPackets function 1908
RTPRssmComputeChunkSize function 1909
RTPRssmCopyDataToChunk function 1909
RTPRssmDecrChunkRefCount function 1910
RTPRssmFillPacketListParams function 1911
RTPRssmGetCapabilities function 1912
RTPRssmGetChunkAndIncrRefCount function 1912
RTPRssmGetExtChunkAndIncrRefCount function 1913
RTPRssmGetInfo function 1914
RTPRssmGetPayloadHeaderLength function 1915
RTPRssmGetStreamHandler function 1916
RTPRssmGetTimeScale function 1916
RTPRssmGetTimeScaleFromPacket function 1917
RTPRssmHandleNewPacket function 1918
RTPRssmHasCharacteristic function 1918
RTPRssmIncrChunkRefCount function 1919
RTPRssmInitialize function 1920
RTPRssmInitParams structure 1951
RTPRssmNewStreamHandler function 1920
RTPRssmPacket structure 1951
RTPRssmReleasePacketList function 1921
RTPRssmReset function 1922
RTPRssmSendChunkAndDecrRefCount function 1923
RTPRssmSendLostChunk function 1923
RTPRssmSendPacketList function 1924
RTPRssmSendPacketList Values 1966
RTPRssmSendStreamBufferRange function 1925
RTPRssmSendStreamHandlerChanged function 1925
RTPRssmSetCapabilities function 1926
RTPRssmSetCapabilities Values 1966
RTPRssmSetInfo function 1927
RTPRssmSetPayloadHeaderLength function 1928
RTPRssmSetSampleDescription function 1928
RTPRssmSetStreamHandler function 1929
RTPRssmSetTimeScale function 1930
RTPSendStreamBufferRangeParams structure 1953

S

SampleNumToMediaDecodeTime function 1638
SampleNumToMediaDisplayTime function 1638
SampleNumToMediaTime function 1639
SampleReference64Ptr data type 1662
SampleReference64Record structure 1662
SampleReferencePtr data type 1663
SampleReferenceRecord structure 1663
ScaleMatrix function 713
scaleMatrixType constant 1958
ScaleMovieSegment function 1640
ScaleTrackSegment function 1641
scaleTranslateMatrixType constant 1958
SCAsyncIdle function 514
SCAudioInvokeLegacyCodecOptionsDialog function 514
scAvailableCompressionListType constant 401, 570
scCodecManufacturerType constant 401, 570
scCodecSettingsType constant 401, 570
SCCompressImage function 514
SCCompressPicture function 515
SCCompressPictureFile function 516
SCCompressSequenceBegin function 517
SCCompressSequenceEnd function 518
SCCompressSequenceFrame function 518
SCCompressSequenceFrameAsync function 520
SCCopyCompressionSessionOptions function 521
scDataRateSettingsType constant 401, 570
SCDefaultPictFileSettings function 522
SCDefaultPictHandleSettings function 522
SCDefaultPixMapSettings function 523
scForceKeyValueType constant 401, 570
SCGetBestDeviceRect function 524
SCGetCompressFlags function 524
scGetCompression constant 572
SCGetCompressionExtended function 525
SCGetInfo function 526
SCGetSettingsAsAtomContainer function 527
SCGetSettingsAsText function 527
SCModalFilterProc callback 553
SCModalFilterUPP data type 556
SCModalHookProc callback 554
SCModalHookUPP data type 556
SCNewGWorld function 528
SCParams structure 557
SCParams Values 571
SCPositionDialog function 529

SCPositionRect function 529
SCRequestImageSettings function 530
SCRequestSequenceSettings function 531
SCSetCompressFlags function 532
SCSetCompressFlags Values 571
SCSetInfo function 532
SCSetSettingsFromAtomContainer function 533
SCSetTestImagePictFile function 534
SCSetTestImagePictHandle function 535
SCSetTestImagePixMap function 536
scShowMotionSettings constant 572
scSoundChannelCountType constant 402, 571
scSoundCompressionType constant 402, 571
scSoundInputSampleRateType constant 402, 571
scSoundSampleRateChangeOK constant 401, 570
scSoundSampleRateType constant 402, 571
scSoundSampleSizeType constant 402, 571
scSoundVBRCompressionOK constant 401, 570
scSpatialSettingsType constant 401, 570
scTemporalSettingsType constant 401, 570
scVideoAllowFrameReorderingType constant 400, 569
scWindowOptionsType constant 401, 570
SelectMovieAlternates function 1642
seqGrabAlwaysUseTimeBase 2247
seqGrabAppendToFile 2248
seqGrabCanMoveWindowWhileRecording 2247
SeqGrabComponent data type 2236
SeqGrabExtendedFrameInfo structure 2237
SeqGrabExtendedFrameInfoPtr data type 2238
SeqGrabFrameInfo structure 2238
SeqGrabFrameInfoPtr data type 2239
SetAutoTrackAlternatesEnabled function 1642
SetCompressedPixMapInfo function 714
SetCSequenceDataRateParams function 715
SetCSequenceFlushProc function 716
SetCSequenceFrameNumber function 717
SetCSequenceKeyFrameRate function 718
SetCSequencePreferredPacketSize function 718
SetCSequencePrev function 719
SetCSequenceQuality function 720
SetDSequenceAccuracy function 721
SetDSequenceDataProc function 722
SetDSequenceFlags function 723
SetDSequenceMask function 724
SetDSequenceMatrix function 724
SetDSequenceMatte function 725
SetDSequenceNonScheduledDisplayDirection function 726
SetDSequenceNonScheduledDisplayTime function 726
SetDSequenceSrcRect function 727
SetDSequenceTimeCode function 728
SetDSequenceTransferMode function 728
SetIdentityMatrix function 729
SetImageDescriptionCTable function 730
SetMediaDataHandler function 1643
SetMediaDataRef function 1474
SetMediaDataRefAttributes function 1475
SetMediaDataRefAttributes Values 1520
SetMediaDefaultDataRefIndex function 1643
SetMediaHandler function 1644
SetMediaInputMap function 1645
SetMediaLanguage function 1646
SetMediaPlayHints function 1475
SetMediaPreferredChunkSize function 1647
SetMediaPropertyAtom function 1476
SetMediaQuality function 1647
SetMediaSampleDescription function 1648
SetMediaShadowSync function 1649
SetMediaTimeScale function 1649
SetMovieActive function 284
SetMovieActiveSegment function 285
SetMovieAnchorDataRef function 1478
SetMovieAudioBalance function 1478
SetMovieAudioContext function 286
SetMovieAudioFrequencyMeteringNumBands function 1479
SetMovieAudioGain function 1480
SetMovieAudioMute function 1480
SetMovieAudioVolumeMeteringEnabled function 1481
SetMovieBox function 286
SetMovieClipRgn function 287
SetMovieColorTable function 1482
SetMovieCoverProcs function 1482
SetMovieDefaultDataRef function 1484
SetMovieDisplayClipRgn function 288
SetMovieDrawingCompleteProc function 289
SetMovieDrawingCompleteProc Values 358
SetMovieGWorld function 290
SetMovieLanguage function 1484
SetMovieMasterClock function 290
SetMovieMasterTimeBase function 291
SetMovieMatrix function 292
SetMoviePlayHints function 1485
SetMoviePosterTime function 293
SetMoviePreferredRate function 293
SetMoviePreferredVolume function 294
SetMoviePreviewMode function 295
SetMoviePreviewTime function 296
SetMovieProgressProc function 1486
SetMoviePropertyAtom function 1487
SetMovieRate function 296
SetMovieSelection function 297
SetMoviesErrorProc function 298

SetMovieTime function 299
SetMovieTimeScale function 300
SetMovieTimeValue function 300
SetMovieVideoOutput function 301
SetMovieVisualBrightness function 1487
SetMovieVisualContext function 302
SetMovieVisualContrast function 1488
SetMovieVisualHue function 1489
SetMovieVisualSaturation function 1489
SetMovieVolume function 302
SetPosterBox function 1490
SetQuickTimePreference function 1491
SetQuickTimePreference Values 1516
SetSequenceProgressProc function 731
SetSpriteProperty function 1491
SetSpriteWorldClip function 1493
SetSpriteWorldFlags function 1494
SetSpriteWorldGraphicsMode function 1494
SetSpriteWorldMatrix function 1495
SetTimeBaseFlags function 303
SetTimeBaseMasterClock function 304
SetTimeBaseMasterTimeBase function 305
SetTimeBaseOffsetTimeBase function 306
SetTimeBaseRate function 306
SetTimeBaseStartTime function 307
SetTimeBaseStopTime function 307
SetTimeBaseTime function 308
SetTimeBaseValue function 309
SetTimeBaseZero function 309
SetTrackAlternate function 1650
SetTrackAudioGain function 1496
SetTrackAudioMute function 1496
SetTrackClipRgn function 310
SetTrackDimensions function 1650
SetTrackEnabled function 1651
SetTrackGWorld function 311
SetTrackLayer function 1652
SetTrackLoadSettings function 1497
SetTrackLoadSettings Values 1522
SetTrackMatrix function 1653
SetTrackMatte function 312
SetTrackOffset function 1653
SetTrackReference function 1654
SetTrackSoundLocalizationSettings function 1655
SetTrackUsage function 1656
SetTrackUsage Values 1666
SetTrackVolume function 1657
SetUserDataItem function 1498
SGAddExtendedFrameReference function 2107
SGAddExtendedMovieData function 2107
SGAddFrame function 2109
SGAddFrameBottleProc callback 2228
SGAddFrameBottleUPP data type 2239
SGAddFrameReference function 2110
SGAddMovieData function 2110
SGAddMovieData Values 2249
SGAddOutputDataRefToMedia function 2111
SGAlignChannelRect function 2112
SGAppendDeviceListToMenu function 2113
SGChangedSource function 2113
SGChannel data type 2239
SGChannelGetCodecSettings function 2114
SGChannelGetDataSourceName function 2115
SGChannelGetRequestedDataRate function 2115
SGChannelPutPicture function 2116
SGChannelSetCodecSettings function 2116
SGChannelSetDataSourceName function 2117
SGChannelSetRequestedDataRate function 2118
SGCompressBottleProc callback 2229
SGCompressBottleUPP data type 2240
SGCompressCompleteBottleProc callback 2230
SGCompressCompleteBottleUPP data type 2240
SGCompressFrame function 2118
SGCompressFrameComplete function 2119
SGCompressInfo structure 2240
SGDataProc callback 2230
SGDataUPP data type 2241
SGDeviceList data type 2241
SGDeviceListPtr data type 2241
SGDisplayBottleProc callback 2231
SGDisplayBottleUPP data type 2242
SGDisplayCompress function 2120
SGDisplayCompressBottleProc callback 2232
SGDisplayCompressBottleUPP data type 2242
SGDisplayFrame function 2121
SGDisposeChannel function 2121
SGDisposeDeviceList function 2122
SGDisposeOutput function 2123
sgFlagAllowNonRGBPixMaps 2249
SGGetAdditionalSoundRates function 2123
SGGetAlignmentProc function 2124
SGGetBufferInfo function 2125
SGGetChannelBounds function 2126
SGGetChannelClip function 2126
SGGetChannelDeviceAndInputNames function 2127
SGGetChannelDeviceList function 2128
SGGetChannelDeviceList Values 2249
SGGetChannelInfo function 2128
SGGetChannelMatrix function 2129
SGGetChannelMaxFrames function 2130
SGGetChannelPlayFlags function 2130
SGGetChannelRefCon function 2131
SGGetChannelSampleDescription function 2132
SGGetChannelSettings function 2132
SGGetChannelTimeBase function 2133
SGGetChannelTimeScale function 2134

SGGetChannelUsage function 2135
SGGetChannelVolume function 2135
SGGetCompressBuffer function 2136
SGGetDataOutput function 2137
SGGetDataOutputStorageSpaceRemaining function 2137
SGGetDataOutputStorageSpaceRemaining64 function 2138
SGGetDataRate function 2139
SGGetDataRef function 2139
SGGetFlags function 2140
SGGetFrameRate function 2141
SGGetGWorld function 2142
SGGetIndChannel function 2142
SGGetInstrument function 2143
SGGetLastMovieResID function 2144
SGGetMaximumRecordTime function 2144
SGGetMode function 2145
SGGetMovie function 2145
SGGetNextExtendedFrameReference function 2146
SGGetNextFrameReference function 2147
SGGetOutputDataReference function 2147
SGGetOutputMaximumOffset function 2148
SGGetOutputNextOutput function 2149
SGGetPause function 2149
SGGetPause Values 2248
SGGetPreferredPacketSize function 2150
SGGetSettings function 2151
SGGetSoundInputDriver function 2151
SGGetSoundInputParameters function 2152
SGGetSoundInputRate function 2153
SGGetSoundRecordChunkSize function 2153
SGGetSrcVideoBounds function 2154
SGGetStorageSpaceRemaining function 2154
SGGetStorageSpaceRemaining64 function 2155
SGGetTextReturnToSpaceValue function 2156
SGGetTimeBase function 2156
SGGetTimeRemaining function 2157
SGGetUserVideoCompressorList function 2157
SGGetUseScreenBuffer function 2158
SGGetVideoBottlenecks function 2159
SGGetVideoCompressor function 2159
SGGetVideoCompressorType function 2161
SGGetVideoDigitizerComponent function 2161
SGGetVideoRect function 2162
SGGrabBottleProc callback 2233
SGGrabBottleUPP data type 2242
SGGrabCompleteBottleProc callback 2233
SGGrabCompleteBottleUPP data type 2242
SGGrabCompressComplete function 2163
SGGrabCompressCompleteBottleProc callback 2234
SGGrabCompressCompleteBottleUPP data type 2242
SGGrabFrame function 2163
SGGrabFrameComplete function 2164
SGGrabPict function 2165
SGGrabPict Values 2247
SGHandleUpdateEvent function 2166
SGIdle function 2166
SGInitChannel function 2167
SGInitialize function 2168
SGModalFilterProc callback 2235
SGModalFilterUPP data type 2243
SGNewChannel function 2169
SGNewChannelFromComponent function 2170
SGNewOutput function 2171
SGOutput data type 2243
SGOutputRecord structure 2243
SGPanelCanRun function 2172
SGPanelEvent function 2173
SGPanelGetDitl function 2174
SGPanelGetDITLForSize function 2175
SGPanelGetDITLForSize Values 402
SGPanelGetSettings function 2176
SGPanelGetTitle function 2177
SGPanelInstall function 2177
SGPanelItem function 2178
SGPanelRemove function 2179
SGPanelSetEventFilter function 2180
SGPanelSetGrabber function 2181
SGPanelSetResFile function 2182
SGPanelSetSettings function 2182
SGPanelValidateInput function 2183
SGPause function 2184
SGPrepare function 2185
SGRelease function 2186
SGSetAdditionalSoundRates function 2186
SGSetChannelBounds function 2187
SGSetChannelClip function 2188
SGSetChannelDevice function 2188
SGSetChannelDeviceInput function 2189
SGSetChannelMatrix function 2190
SGSetChannelMaxFrames function 2190
SGSetChannelOutput function 2191
SGSetChannelPlayFlags function 2192
SGSetChannelRefCon function 2192
SGSetChannelSettings function 2193
SGSetChannelSettingsStateChanging function 2194
SGSetChannelSettingsStateChanging Values 2249
SGSetChannelUsage function 2195
SGSetChannelVolume function 2195
SGSetCompressBuffer function 2196
SGSetDataOutput function 2197
SGSetDataProc function 2198
SGSetDataRef function 2199
SGSetFlags function 2200
SGSetFontName function 2200

SGSetFontSize function 2201
SGSetFrameRate function 2202
SGSetGWorld function 2202
SGSetInstrument function 2204
SGSetJustification function 2204
SGSetMaximumRecordTime function 2205
SGSetOutputFlags function 2205
SGSetOutputMaximumOffset function 2206
SGSetOutputNextOutput function 2207
SGSetPreferredPacketSize function 2208
SGSetSettings function 2208
SGSetSettingsSummary function 2209
SGSetSoundInputDriver function 2210
SGSetSoundInputParameters function 2210
SGSetSoundInputRate function 2211
SGSetSoundRecordChunkSize function 2211
SGSetTextBackColor function 2212
SGSetTextForeColor function 2213
SGSetTextReturnToSpaceValue function 2213
SGSettingsDialog function 2214
SGSettingsDialog Values 2248
SGSetUserVideoCompressorList function 2215
SGSetUseScreenBuffer function 2216
SGSetVideoBottlenecks function 2217
SGSetVideoCompressor function 2218
SGSetVideoCompressorType function 2219
SGSetVideoDigitizerComponent function 2219
SGSetVideoRect function 2220
SGSortDeviceList function 2221
SGSoundInputDriverChanged function 2221
SGStartPreview function 2222
SGStartRecord function 2222
SGStop function 2223
SGTransferFrameBottleProc callback 2235
SGTransferFrameBottleUPP data type 2243
SGTransferFrameForCompress function 2223
SGUpdate function 2224
SGVideoDigitizerChanged function 2225
SGWriteExtendedMovieData function 2226
SGWriteMovieData function 2227
SGWriteSamples function 2227
SHChunkRecord structure 1954
SHExtendedChunkRecord structure 1955
SHExtendedChunkRecord Values 1966
ShowMovieInformation function 1499
ShowMoviePoster function 312
SHServerEditParameters structure 1956
SkewMatrix function 731
smCentralEuroRoman constant 440
SMPTEFlags data type 956
SMPTEFrameReference data type 957
SMPTEWipeType data type 957
smRSymbol constant 440
smSimpChinese constant 440
smTradChinese constant 440
SoundMediaType constant 403, 1667
Sprite data type 1514
Sprite Properties 387, 1519
SpriteHitTest function 1500
SpriteMediaCountImages function 313
SpriteMediaCountSprites function 314
SpriteMediaDisposeImage function 314
SpriteMediaDisposeSprite function 315
SpriteMediaGetActionVariable function 315
SpriteMediaGetActionVariableAsString function 316
SpriteMediaGetDisplayedSampleNumber function 317
SpriteMediaGetImageName function 317
SpriteMediaGetIndImageDescription function 318
SpriteMediaGetIndImageProperty function 319
SpriteMediaGetProperty function 319
SpriteMediaGetSpriteActionsForQTEvent function 320
SpriteMediaGetSpriteName function 321
SpriteMediaGetSpriteProperty function 322
SpriteMediaHitTestAllSprites function 323
SpriteMediaHitTestOneSprite function 324
SpriteMediaHitTestSprites function 324
SpriteMediaImageIDToIndex function 325
SpriteMediaImageIndexToID function 326
SpriteMediaNewImage function 326
SpriteMediaNewSprite function 327
SpriteMediaSetActionVariable function 328
SpriteMediaSetActionVariableToString function 328
SpriteMediaSetProperty function 329
SpriteMediaSetSpriteProperty function 330
SpriteMediaSpriteIDToIndex function 331
SpriteMediaSpriteIndexToID function 331
SpriteRecord structure 1515
SpriteWorld data type 1515
SpriteWorldHitTest function 1501
SpriteWorldHitTest Values 403
SpriteWorldIdle function 1502
SpriteWorldIdle Values 1522
SpriteWorldRecord structure 1515
srcBic constant 431
srcCopy constant 430
srcOr constant 431
srcXor constant 431
Standard Compression Constants 398, 567
StartMovie function 332
StdPix function 732
StdPix Values 748
StdPixProc callback 135

StdPixUPP data type 140
StopMovie function 333
Str31 data type 1772
Streaming Transport Atoms 1964
Style data type 355
subOver constant 433
subPin constant 433
SubtractTime function 333
SynthesizerConnections structure 1772
SynthesizerDescription structure 1773

T

tc24HourMax constant 573
tcDropFrame constant 572
TCFrameNumberToTimeCode function 538
TCGetCurrentTimeCode function 538
TCGetDisplayOptions function 539
TCGetSourceRef function 540
TCGetTimeCodeAtTime function 540
TCGetTimeCodeFlags function 541
tcNegTimesOK constant 573
TCSetDisplayOptions function 542
TCSetSourceRef function 543
TCSetTimeCodeFlags function 543
TCSetTimeCodeFlags Values 572
TCTextOptions structure 558
TCTextOptionsPtr data type 559
TCTimeCodeToFrameNumber function 544
TCTimeCodeToString function 545
TEHandle data type 355
TEPtr data type 355
TerminateQTS function 1930
Text Properties 404
TextDescriptionHandle data type 355
TextDescriptionPtr data type 356
TextDisplayData structure 559
TextExportComponent data type 561
TextExportGetDisplayData function 546
TextExportGetSettings function 546
TextExportGetTimeFraction function 547
TextExportSetSettings function 548
TextExportSetSettings Values 565
TextExportSetTimeFraction function 548
TextMediaAddHiliteSample function 334
TextMediaAddTESample function 335
TextMediaAddTextSample function 337
TextMediaDrawRaw function 339
TextMediaFindNextText function 340
TextMediaFindNextText Values 356
TextMediaGetTextProperty function 341
TextMediaHiliteTextSample function 342
TextMediaRawIdle function 343
TextMediaRawSetup function 344
TextMediaSetTextProc function 345
TextMediaSetTextProperty function 345
TextMediaSetTextSampleData function 346
TextMediaType constant 403, 1667
TIFF Properties 1083
timeBaseAfterStopTime 358
TimeBaseStatus data type 356
TimeCodeDef structure 561
TimeCodeDef Values 572
TimeCodeDescriptionHandle data type 562
TimeCodeDescriptionPtr data type 563
TimeCodeRecord structure 563
ToneDescription Values 408
TrackEditState data type 1664
TrackEditStateRecord structure 1664
TrackTimeToMediaDisplayTime function 1658
TrackTimeToMediaTime function 1658
TransformFixedPoints function 734
TransformFixedRect function 734
TransformPoints function 735
TransformRect function 736
TransformRgn function 737
TranslateMatrix function 737
translateMatrixType constant 1957
transparent constant 434
TrimImage function 738
TuneCallBackProc callback 1762
TuneCallBackUPP data type 1776
TuneGetIndexedNoteChannel function (Deprecated in Mac OS X v10.5) 1746
TuneGetNoteAllocator function (Deprecated in Mac OS X v10.5) 1746
TuneGetPartMix function (Deprecated in Mac OS X v10.5) 1747
TuneGetStatus function (Deprecated in Mac OS X v10.5) 1748
TuneGetTimeBase function (Deprecated in Mac OS X v10.5) 1748
TuneGetTimeScale function (Deprecated in Mac OS X v10.5) 1749
TuneGetVolume function (Deprecated in Mac OS X v10.5) 1749
TuneInstant function (Deprecated in Mac OS X v10.5) 1750
TunePlayCallBackProc callback 1763
TunePlayCallBackUPP data type 1776
TunePlayer data type 1776
TunePreroll function (Deprecated in Mac OS X v10.5) 1751
TuneQueue function (Deprecated in Mac OS X v10.5) 1751

TuneSetBalance function (Deprecated in Mac OS X v10.5) 1752
TuneSetHeader function (Deprecated in Mac OS X v10.5) 1753
TuneSetHeaderWithSize function (Deprecated in Mac OS X v10.5) 1754
TuneSetNoteChannels function (Deprecated in Mac OS X v10.5) 1755
TuneSetPartMix function (Deprecated in Mac OS X v10.5) 1755
TuneSetPartMix Values 1788
TuneSetPartTranspose function (Deprecated in Mac OS X v10.5) 1756
TuneSetSofter function (Deprecated in Mac OS X v10.5) 1757
TuneSetSoundLocalization function (Deprecated in Mac OS X v10.5) 1758
TuneSetTimeScale function (Deprecated in Mac OS X v10.5) 1758
TuneSetVolume function (Deprecated in Mac OS X v10.5) 1759
TuneStatus structure 1776
TuneStop function (Deprecated in Mac OS X v10.5) 1759
TuneTask function (Deprecated in Mac OS X v10.5) 1760
TuneUnroll function (Deprecated in Mac OS X v10.5) 1760
TweenerComponent data type 564
TweenerDataProc callback 1508
TweenerDataUPP data type 1516
TweenerDoTween function 549
TweenerInitialize function 550
TweenerReset function 551
TweenRecord structure 564

U

UnsignedFixMulDiv function 739
UpdateMovie function 347
UpdateMovieInStorage function 1503
UpdateMovieResource function 1503
UseMovieEditState function 1659
UseTrackEditState function 1660

V

VDAddKeyColor function 2272
VDCaptureStateChanging function 2272
VDClearClipRgn function 2273
VDCompressDone function 2273
VDCompressionListHandle data type 2347
VDCompressionListPtr data type 2348
VDCompressOneFrameAsync function 2274
VDDone function 2275
vdFlagCaptureAlwaysUseTimeBase 2358
VDGammaRecord structure 2348
VDGamRecPtr data type 2348
VDGetActiveSrcRect function 2276
VDGetBlackLevelValue function 2276
VDGetBrightness function 2277
VDGetClipState function 2278
VDGetCLUTInUse function 2278
VDGetCompressionTime function 2279
VDGetCompressionTypes function 2280
VDGetContrast function 2281
VDGetCurrentFlags function 2281
VDGetDataRate function 2282
VDGetDeviceNameAndFlags function 2283
VDGetDeviceNameAndFlags Values 2358
VDGetDigitizerInfo function 2284
VDGetDigitizerRect function 2285
VDGetDMADepths function 2285
VDGetFieldPreference function 2286
VDGetHue function 2287
VDGetImageDescription function 2288
VDGetInput function 2288
VDGetInputColorSpaceMode function 2289
VDGetInputFormat function 2289
VDGetInputGammaRecord function 2290
VDGetInputGammaValue function 2291
VDGetInputName function 2292
VDGetKeyColor function 2292
VDGetKeyColorRange function 2293
VDGetMaskandValue function 2294
VDGetMaskPixMap function 2295
VDGetMaxAuxBuffer function 2295
VDGetMaxSrcRect function 2296
VDGetNextKeyColor function 2296
VDGetNumberOfInputs function 2297
VDGetPlayThruDestination function 2298
VDGetPLLFilterType function 2299
VDGetPreferredImageDimensions function 2299
VDGetPreferredTimeScale function 2300
VDGetSaturation function 2300
VDGetSharpness function 2301
VDGetSoundInputDriver function 2302
VDGetSoundInputSource function 2302
VDGetTimeCode function 2303
VDGetUniqueIDs function 2304
VDGetVBlankRect function 2304
VDGetVideoDefaults function 2305
VDGetWhiteLevelValue function 2307
VDGrabOneFrame function 2307
VDGrabOneFrameAsync function 2308

VdigBufferRecListHandle data type 2348
VdigBufferRecListPtr data type 2349
VdigType Values 2359
VDIIDCGetCSRData function 2309
VDIIDCGetDefaultFeatures function 2310
VDIIDCGetFeatures function 2310
VDIIDCGetFeaturesForSpecifier function 2311
VDIIDCSetCSRData function 2312
VDIIDCSetFeatures function 2313
VDPreflightDestination function 2314
VDPreflightGlobalRect function 2315
VDReleaseAsyncBuffers function 2316
VDReleaseCompressBuffer function 2316
VDResetCompressSequence function 2317
VDSelectUniqueIDs function 2317
VDSetBlackLevelValue function 2318
VDSetBrightness function 2318
VDSetClipRgn function 2319
VDSetClipState function 2320
VDSetCompression function 2320
VDSetCompression Values 379
VDSetCompressionOnOff function 2321
VDSetContrast function 2322
VDSetDataRate function 2323
VDSetDestinationPort function 2323
VDSetDigitizerRect function 2324
VDSetDigitizerUserInterrupt function 2324
VDSetFieldPreference function 2325
VDSetFieldPreference Values 2359
VDSetFrameRate function 2326
VDSetHue function 2326
VDSetInput function 2327
VDSetInputColorSpaceMode function 2328
VDSetInputGammaRecord function 2328
VDSetInputGammaValue function 2329
VDSetInputStandard function 2329
VDSetKeyColor function 2330
VDSetKeyColorRange function 2331
VDSetMasterBlendLevel function 2331
VDSetPlayThruDestination function 2332
VDSetPlayThruGlobalRect function 2333
VDSetPlayThruOnOff function 2334
VDSetPlayThruOnOff Values 2358
VDSetPLLFilterType function 2334
VDSetPreferredImageDimensions function 2335
VDSetPreferredPacketSize function 2335
VDSetSaturation function 2336
VDSetSharpness function 2337
VDSetTimeBase function 2337
VDSetupBuffers function 2338
VDSetWhiteLevelValue function 2338
vdTypeAlpha constant 2359
vdTypeBasic constant 2359
vdTypeMask constant 2359
VDUseSafeBuffers function 2339
VDUseThisCLUT function 2340
verChina constant 440
verFarEastGeneric constant 440
verGreece constant 440
verGreecePoly constant 441
verInternational constant 441
verIreland constant 439
verIrishGaelicScript constant 439
verMultilingual constant 441
verScriptGeneric constant 441
verSpain constant 441
verSpLatinAmerica constant 441
verTaiwan constant 440
Video Digitizer Capabilities 2350
VideoBottles structure 2244
VideoDigitizerComponent data type 2349
VideoDigitizerError data type 2349
VideoMediaGetCodecParameter function 348
VideoMediaGetStallCount function 349
VideoMediaGetStatistics function 349
VideoMediaResetStatistics function 350
VideoMediaSetCodecParameter function 351

X

XMLDoc data type 845
XMLDocRecord structure 846
XMLParseAddAttribute function 829
XMLParseAddAttributeAndValue function 830
XMLParseAddAttributeValueKind function 830
XMLParseAddElement function 831
XMLParseAddMultipleAttributes function 832
XMLParseAddMultipleAttributesAndValues function 833
XMLParseAddNameSpace function 833
XMLParseDataRef function 834
XMLParseDisposeXMLDoc function 835
XMLParseFile function 835
XMLParseGetDetailedParseError function 836
XMLParseSetCDataHandler function 836
XMLParseSetCharDataHandler function 837
XMLParseSetCommentHandler function 837
XMLParseSetEndDocumentHandler function 838
XMLParseSetEndElementHandler function 838
XMLParseSetEventParseRefCon function 839
XMLParseSetOffsetAndLimit function 839
XMLParseSetPreprocessInstructionHandler function 840
XMLParseSetStartDocumentHandler function 841
XMLParseSetStartElementHandler function 841