Movie Toolkit Reference

QuickTime > Movie Creation

2006-05-23

Apple Inc.
© 2006 Apple Computer, Inc.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Inc., with the following exceptions: Any person is hereby authorized to store documentation on a single computer for personal use only and to print copies of documentation for personal use provided that the documentation contains Apple's copyright notice.

The Apple logo is a trademark of Apple Inc.

Use of the "keyboard" Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

No licenses, express or implied, are granted with respect to any of the technology described in this document. Apple retains all intellectual property rights associated with the technology described in this document. This document is intended to assist application developers to develop applications only for Apple-labeled computers.

Every effort has been made to ensure that the information in this document is accurate. Apple is not responsible for typographical errors.

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

Apple, the Apple logo, Carbon, Cocoa, Mac, Mac OS, Macintosh, Quartz, QuickDraw, QuickTime, and SoundTrack are trademarks of Apple Inc., registered in the United States and other countries.

OpenGL is a registered trademark of Silicon Graphics, Inc.

PowerPC and and the PowerPC logo are trademarks of International Business Machines Corporation, used under license therefrom.

Times is a registered trademark of Heidelberger Druckmaschinen AG, available from Linotype Library GmbH.

Simultaneously published in the United States and Canada.

Even though Apple has reviewed this document, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS DOCUMENT IS PROVIDED "AS IS," AND YOU, THE READER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Contents

2006-05-23 | © 2006 Apple Computer, Inc. All Rights Reserved.

Movie Toolkit Reference

Framework:	Frameworks/QuickTime.framework
Declared in	Files.h IOHIDDescriptorParser.h 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 51)
    Disposes of a movie controller.

`NewMovieController` (page 119)
    Locates a movie controller component and assigns a movie to that controller.

Audio Conversion and Extraction

`MovieAudioExtractionBegin` (page 111)
    Begins a movie audio extraction session.

`MovieAudioExtractionEnd` (page 111)
    Ends a movie audio extraction session.

`MovieAudioExtractionFillBuffer` (page 112)
    Extracts audio from a movie.

`MovieAudioExtractionGetProperty` (page 113)
    Gets a property of a movie audio extraction session.

`MovieAudioExtractionGetPropertyInfo` (page 114)
    Gets information about a property of a movie audio extraction session.

`MovieAudioExtractionSetProperty` (page 115)
    Sets a property of a movie audio extraction session.

Copying Existing Atoms

`QTCopyAtom` (page 148)
Copies an atom and its children to a new atom container.

`QTInsertChildren` (page 174)
Inserts a container of atoms as children of the specified parent atom.

`QTReplaceAtom` (page 190)
Replaces the contents of an atom and its children with a different atom and its children.

`QTSwapAtoms` (page 196)
Swaps the contents of two atoms in an atom container.

Creating and Disposing of Atom Containers

`QTDisposeAtomContainer` (page 154)
Disposes of an atom container.

`QTNewAtomContainer` (page 178)
Creates a new atom container.

Creating and Manipulating Sprites

`DisposeSprite` (page 59)
Disposes of a sprite.

`GetSpriteProperty` (page 98)
Retrieves the value of a specified sprite property.

`InvalidateSprite` (page 107)
Invalidates the portion of a sprite's sprite world that is occupied by a sprite.

`NewSprite` (page 137)
Creates a new sprite in a specified sprite world.

`SetSpriteProperty` (page 218)
Sets the specified property of a sprite.

`SpriteHitTest` (page 227)
Determines whether a location in a sprite's display coordinate system intersects the sprite.

Creating New Atoms

`QTInsertChild` (page 173)
Creates a new child atom of the specified parent atom.

Enhancing Movie Playback Performance

`GetTrackLoadSettings` (page 100)
Retrieves a track's preload information.

`SetMediaPlayHints` (page 202)
: Provides information to the Movie Toolbox that can influence playback of a single media.

`SetMoviePlayHints` (page 212)
: Provides information to the Movie Toolbox that can influence movie playback.

`SetTrackLoadSettings` (page 224)
: Specifies a portion of a track that is to be loaded into memory whenever it is played.

Error Functions

`QTAddMovieError` (page 147)
: Adds orthogonal errors to a movie's list of errors.

Finding and Adding Samples

`GetMediaNextInterestingDecodeTime` (page 72)
: Searches for decode times of interest in a media.

`GetMediaNextInterestingDisplayTime` (page 73)
: Searches for display times of interest in a media.

Finding Interesting Times

`GetMediaNextInterestingTime` (page 74)
: Searches for times of interest in a media.

`GetMovieNextInterestingTime` (page 87)
: Searches for times of interest in a movie's enabled tracks.

`GetTrackNextInterestingTime` (page 101)
: Searches for times of interest in a track.

High-Level Download Control

`GetMaxLoadedTimeInMovie` (page 69)
: When a movie is being progressively downloaded, returns the duration of the part of a movie that has already been downloaded.

`QTMovieNeedsTimeTable` (page 177)
: Returns whether a movie is being progressively downloaded.

High-Level Effects Functions

`QTCreateStandardParameterDialog` (page 151)
: Creates a dialog box that allows the user to choose an effect from the list of effects passed to the function.

`QTDismissStandardParameterDialog` (page 153)
: Closes a standard parameter dialog box that was created using QTCreateStandardParameterDialog.

`QTGetEffectsList` (page 168)
: Returns a QT atom container holding a list of the currently installed effects components.

`QTGetEffectsListExtended` (page 169)
: Provides for more advanced filtering of effects to be placed into the effect list.

`QTGetEffectSpeed` (page 170)
: Returns the speed of the effect, expressed in frames per second.

`QTIsStandardParameterDialogEvent` (page 175)
: Determines if a Macintosh event is processed by a standard parameter dialog box created by QTCreateStandardParameterDialog.

`QTStandardParameterDialogDoAction` (page 194)
: Lets you change some of the default behaviors of the standard parameter dialog box.

High-Level Movie Editing Functions

`NewMovieFromScrap` (page 129)
: Creates a movie from the contents of the scrap.

`PutMovieOnScrap` (page 146)
: Places a movie into the Macintosh scrap.

Low-Level Download Control

`MakeMediaTimeTable` (page 108)
: Returns a time table for the specified media.

`MakeTrackTimeTable` (page 109)
: Returns a time table for a specified track in a movie.

Metering Sound Level and Frequency

`GetMovieAudioFrequencyLevels` (page 79)
: Returns the current frequency meter levels of a movie mix.

`GetMovieAudioFrequencyMeteringBandFrequencies` (page 79)
: Returns the chosen middle frequency for each band in the configured frequency metering of a particular movie mix.

`GetMovieAudioFrequencyMeteringNumBands` (page 80)
: Returns the number of frequency bands being metered for a movie's specified audio mix.

`GetMovieAudioVolumeMeteringEnabled` (page 83)
: Returns the enabled or disabled status of volume metering of a particular audio mix of a movie.

`SetMovieAudioFrequencyMeteringNumBands` (page 206)
: Configures frequency metering for a particular audio mix in a movie.

`SetMovieAudioVolumeMeteringEnabled` (page 208)
: Enables or disables volume metering of a particular audio mix of a movie.

Modifying Atoms

`QTSetAtomData` (page 192)
: Changes the data of a leaf atom.

`QTSetAtomID` (page 194)
: Changes the ID of an atom.

Movie Functions

`CloseMovieFile` (page 39)
: Closes an open movie file.

`CreateMovieFile` (page 44)
: Creates a movie file, creates an empty movie which references the file, and opens the movie file with write permission.

`DeleteMovieFile` (page 48)
: Deletes a movie file.

`NewMovieForDataRefFromHandle` (page 121)
: Creates a movie from a public movie handle, converting internal references to external references.

`NewMovieFromDataRef` (page 124)
: Creates a movie from any device with a corresponding data handler.

`NewMovieFromFile` (page 126)
: 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 128)
: Creates a movie in memory from a movie resource or a handle you obtained from PutMovieIntoHandle.

`NewMovieFromUserProc` (page 131)
: Creates a movie from data that you provide.

`OpenMovieFile` (page 143)
: Opens a specified movie file.

Movie Posters and Movie Previews

`GetPosterBox` (page 95)
: Obtains a poster's boundary rectangle.

`SetPosterBox` (page 217)
: Sets a poster's boundary rectangle.

Movies and Your Event Loop

`DisposeQTNextTaskNeededSoonerCallbackUPP` (page 58)
: Disposes of a QTNextTaskNeededSoonerCallbackUPP pointer.

`GetMovieStatus` (page 91)
: 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.

2006-05-23 | © 2006 Apple Computer, Inc. All Rights Reserved.

`GetTrackStatus` (page 103)
: Returns the value of the last error the media encountered while playing a specified track.

`NewQTNextTaskNeededSoonerCallbackUPP` (page 136)
: Allocates a Universal Procedure Pointer for the QTNextTaskNeededSoonerCallbackProc callback.

Registering and Unregistering Access Keys

`QTRegisterAccessKey` (page 188)
: Registers an access key.

`QTUnregisterAccessKey` (page 197)
: Removes a previously registered access key.

Removing Atoms From an Atom Container

`QTRemoveAtom` (page 188)
: Removes an atom and its children from the specified atom container.

`QTRemoveChildren` (page 189)
: Removes all the children of an atom from the specified atom container.

Retrieving Access Keys

`QTGetAccessKeys` (page 159)
: Returns all the application and system access keys of a specified access key type.

Retrieving Atoms and Atom Data

`QTCopyAtomDataToHandle` (page 149)
: Copies the specified leaf atom's data to a handle.

`QTCopyAtomDataToPtr` (page 150)
: Copies the specified leaf atom's data to a buffer.

`QTCountChildrenOfType` (page 151)
: Returns the number of atoms of a given type in the child list of the specified parent atom.

`QTFindChildByID` (page 157)
: Retrieves an atom by ID from the child list of the specified parent atom.

`QTFindChildByIndex` (page 158)
: Retrieves an atom by index from the child list of the specified parent atom.

`QTGetAtomDataPtr` (page 160)
: Retrieves a pointer to the atom data for a specified leaf atom.

`QTGetAtomTypeAndID` (page 162)
: Retrieves an atom's type and ID.

`QTGetNextChildType` (page 172)
: Returns the next atom type in the child list of the specified parent atom.

`QTLockContainer` (page 176)
  Locks an atom container in memory.

`QTNextChildAnyType` (page 187)
  Returns the next atom in the child list of the specified parent atom.

`QTUnlockContainer` (page 196)
  Unlocks an atom container in memory.

Saving Movies

`AddMovieResource` (page 27)
  Adds a movie resource to a specified resource file.

`AddMovieToStorage` (page 29)
  Adds a movie to a storage container that was created by CreateMovieStorage.

`ClearMovieChanged` (page 38)
  Sets the movie changed flag to indicate that the movie has not been changed.

`CloseMovieStorage` (page 40)
  Closes an open movie storage container.

`CreateMovieStorage` (page 46)
  Creates an empty storage location to hold a movie and opens a data handler to the stored movie with write permission.

`DeleteMovieStorage` (page 49)
  Deletes a movie storage container.

`FlattenMovie` (page 64)
  Creates a new movie file containing a specified movie.

`FlattenMovieData` (page 66)
  Creates a new movie and a file that contains all the movie data.

`FlattenMovieDataToDataRef` (page 68)
  Performs a flattening operation to a movie at a storage location.

`HasMovieChanged` (page 106)
  Determines whether a movie has changed and needs to be saved.

`NewMovieFromDataFork` (page 122)
  Retrieves a movie that is stored anywhere in the data fork of a specified Macintosh file.

`NewMovieFromStorageOffset` (page 130)
  Creates a new movie based on the offset to data in a storage container.

`RemoveMovieResource` (page 198)
  Removes a movie resource from a specified movie file.

`UpdateMovieInStorage` (page 230)
  Updates a movie at a storage location.

`UpdateMovieResource` (page 230)
  Replaces the contents of a movie resource in a specified movie file.

Setting Sound Parameters

`GetMovieAudioBalance` (page 78)
: Returns the balance value for the audio mix of a movie currently playing.

`GetMovieAudioGain` (page 81)
: Returns the gain value for the audio mix of a movie currently playing.

`GetTrackAudioGain` (page 98)
: Returns the gain value for the audio mix of a track currently playing.

`GetTrackAudioMute` (page 99)
: Returns the mute value for the audio mix of a track currently playing.

`SetMovieAudioBalance` (page 205)
: Sets the balance level for the mixed audio output of a movie.

`SetMovieAudioGain` (page 207)
: Sets the audio gain level for the mixed audio output of a movie, altering the perceived volume of the movie's playback.

`SetMovieAudioMute` (page 207)
: Sets the mute value for the audio mix of a movie currently playing.

`SetTrackAudioGain` (page 223)
: Sets the audio gain level for the audio output of a track, altering the perceived volume of the track's playback.

`SetTrackAudioMute` (page 223)
: Mutes or unmutes the audio output of a track.

Tween Component Requirements

`QTDoTweenPtr` (page 156)
: Runs a tween component and returns values in a pointer rather than a handle.

Using the Full Screen

`BeginFullScreen` (page 33)
: Begins full-screen mode for a specified graphics device.

`EndFullScreen` (page 63)
: Ends full-screen mode for a graphics device.

Working With Alternate Tracks

`SetMovieLanguage` (page 211)
: Specifies a movie's localized language or region code.

Working With Data References

`AddMediaDataRef` (page 26)
: Adds a data reference to a media.

`GetMediaDataRef` (page 70)
: Returns a copy of a specified data reference.

`GetMediaDataRefCount` (page 71)
: Determines the number of data references in a media.

Working With Media Handler Properties

`GetMediaPropertyAtom` (page 76)
: Retrieves the property atom container of a media handler.

`SetMediaPropertyAtom` (page 203)
: Sets the property atom container of a media handler.

Working With Movie Restrictions

`QTCreateUUID` (page 153)
: Creates a 128-bit universal unique ID number.

`QTEqualUUIDs` (page 157)
: Compares two 128-bit ID numbers.

`QTGetMovieRestrictions` (page 171)
: Returns the restrictions, if any, for a given movie.

`QTGetSupportedRestrictions` (page 172)
: Reports the movie restrictions enforced by the currently running version of QuickTime.

`QTRestrictionsGetIndClass` (page 190)
: Reports the class of a movie restriction.

`QTRestrictionsGetInfo` (page 191)
: Reports information about the restrictions in a specified restriction set.

`QTRestrictionsGetItem` (page 192)
: Retrieves specific movie restrictions.

Working With Movie Spatial Characteristics

`GetMovieColorTable` (page 84)
: Retrieves a movie's color table.

`GetMovieSegmentDisplayBoundsRgn` (page 90)
: Determines a movie's display boundary region for a specified segment.

`GetTrackSegmentDisplayBoundsRgn` (page 102)
: Determines the region a track occupies in a movie's graphics world during a specified segment.

`SetMovieColorTable` (page 209)
: Associates a ColorTable structure with a movie.

2006-05-23 | © 2006 Apple Computer, Inc. All Rights Reserved.

Working With Progress and Cover Functions

`GetMovieCoverProcs` (page 84)
Retrieves the cover functions that you set with the SetMovieCoverProcs function.

`SetMovieCoverProcs` (page 209)
Sets the callbacks invoked when a movie is covered or uncovered.

`SetMovieProgressProc` (page 213)
Attaches a progress function to a movie.

Working With Sprite Worlds

`DisposeAllSprites` (page 50)
Disposes of all sprites associated with a sprite world.

`DisposeSpriteWorld` (page 60)
Disposes of a sprite world.

`InvalidateSpriteWorld` (page 107)
Invalidates a rectangular area of a sprite world.

`NewSpriteWorld` (page 139)
Creates a new sprite world.

`SetSpriteWorldClip` (page 220)
Sets a sprite world's clip shape to the specified region.

`SetSpriteWorldMatrix` (page 222)
Sets a sprite world's matrix to the specified matrix.

`SpriteWorldHitTest` (page 228)
Determines whether any sprites are at a specified location in a sprite world.

`SpriteWorldIdle` (page 229)
Allows a sprite world to update its invalid areas.

Working With User Data

`AddUserData` (page 30)
Adds an item to a user data list.

`AddUserDataText` (page 31)
Places language-tagged text into an item in a user data list.

`CopyMediaUserData` (page 40)
Copies a source media's user data into a destination media's user data.

`CopyMovieUserData` (page 41)
Copies a source movie's user data into a destination movie's user data.

`CopyTrackUserData` (page 42)
Copies a source track's user data into a destination track's user data.

`CopyUserData` (page 42)
Copies metadata items from the source user data container to the destination user data container.

`CountUserDataType` (page 43)
: Determines the number of items of a given type in a user data list.

`DisposeUserData` (page 63)
: Disposes of a user data structure created by NewUserData.

`GetNextUserDataType` (page 94)
: Retrieves the next user data type in a specified user data list.

`GetUserData` (page 103)
: Returns a specified user data item.

`GetUserDataItem` (page 104)
: Returns a specified user data item.

`GetUserDataText` (page 105)
: Retrieves language-tagged text from an item in a user data list.

`NewUserData` (page 142)
: Creates a new user data structure.

`NewUserDataFromHandle` (page 143)
: Creates a new user data structure from a handle.

`PutUserDataIntoHandle` (page 147)
: Returns a handle to a user data structure.

`RemoveUserData` (page 199)
: Removes an item from a user data list.

`RemoveUserDataText` (page 200)
: Removes language-tagged text from an item in a user data list.

`SetUserDataItem` (page 225)
: Sets an item in a user data list.

Supporting Functions

`AddMovieExecuteWiredActionsProc` (page 26)
: Lets you add a callback to a movie to execute wired actions.

`AddSoundDescriptionExtension` (page 30)
: Adds an extension to a SoundDescription structure.

`AttachMovieToCurrentThread` (page 32)
: Attaches a movie to the current thread.

`CanQuickTimeOpenDataRef` (page 35)
: Determines whether referenced data can be opened using a graphics importer or opened in place as a movie.

`CanQuickTimeOpenFile` (page 37)
: Determines whether a file can be opened using a graphics importer or opened in place as a movie.

`CreateShortcutMovieFile` (page 47)
: Creates a movie file that just contains a reference to another movie.

`DetachMovieFromCurrentThread` (page 49)
: Detaches a movie from the current thread.

`DisposeActionsUPP` (page 50)
: Disposes of an ActionsUPP pointer.

`DisposeDoMCActionUPP` (page 51)
  Disposes of a DoMCActionUPP pointer.

`DisposeGetMovieUPP` (page 51)
  Disposes of a GetMovieUPP pointer.

`DisposeMovieDrawingCompleteUPP` (page 54)
  Disposes of a MovieDrawingCompleteUPP pointer.

`DisposeMovieExecuteWiredActionsUPP` (page 55)
  Disposes of a MovieExecuteWiredActionsUPP pointer.

`DisposeMoviePrePrerollCompleteUPP` (page 55)
  Disposes of a MoviePrePrerollCompleteUPP pointer.

`DisposeMoviePreviewCallOutUPP` (page 56)
  Disposes of a MoviePreviewCallOutUPP pointer.

`DisposeMovieProgressUPP` (page 56)
  Disposes of a MovieProgressUPP pointer.

`DisposeMovieRgnCoverUPP` (page 57)
  Disposes of a MovieRgnCoverUPP pointer.

`DisposeMoviesErrorUPP` (page 57)
  Disposes of a MoviesErrorUPP pointer.

`DisposeQTCallBackUPP` (page 57)
  Disposes of a QTCallBackUPP pointer.

`DisposeQTEffectListFilterUPP` (page 58)
  Disposes of a QTEffectListFilterUPP pointer.

`DisposeQTSyncTaskUPP` (page 59)
  Disposes of a QTSyncTaskUPP pointer.

`DisposeTextMediaUPP` (page 61)
  Disposes of a TextMediaUPP pointer.

`DisposeTrackTransferUPP` (page 62)
  Disposes of a TrackTransferUPP pointer.

`DisposeTweenerDataUPP` (page 62)
  Disposes of a TweenerDataUPP pointer.

`GetMediaPlayHints` (page 76)
  Undocumented

`GetMovieAnchorDataRef` (page 77)
  Retrieves a movie's anchor data reference and type.

`GetMovieAudioMute` (page 81)
  Returns the mute value for the audio mix of a movie currently playing.

`GetMovieAudioVolumeLevels` (page 82)
  Returns the current volume meter levels of a movie.

`GetMovieDefaultDataRef` (page 85)
  Gets a movie's default data reference.

`GetMovieLoadState` (page 86)
  Returns a value that indicates the state of a movie's loading process.

`GetMovieProgressProc` (page 89)
  Gets the MovieProgressProc callback attached to a movie.

`GetMoviePropertyAtom` (page 89)
: Gets a movie's property atom.

`GetMovieThreadAttachState` (page 91)
: Determines whether a given movie is attached to a thread.

`GetMovieVisualBrightness` (page 92)
: Returns the brightness adjustment for the movie.

`GetMovieVisualContrast` (page 92)
: Returns the contrast adjustment for the movie.

`GetMovieVisualHue` (page 93)
: Returns the hue adjustment for the movie.

`GetMovieVisualSaturation` (page 94)
: Returns the color saturation adjustment for the movie.

`GetQuickTimePreference` (page 96)
: Retrieves a particular preference from the QuickTime preferences.

`GetSoundDescriptionExtension` (page 97)
: Gets the current extension to a SoundDescription structure.

`MovieExecuteWiredActions` (page 116)
: Undocumented

`MovieSearchText` (page 116)
: Searches for text in a movie.

`NewActionsUPP` (page 118)
: Allocates a Universal Procedure Pointer for ActionsProc.

`NewDoMCActionUPP` (page 118)
: Allocates a Universal Procedure Pointer for the DoMCActionProc callback.

`NewGetMovieUPP` (page 119)
: Allocates a Universal Procedure Pointer for the GetMovieProc callback.

`NewMovieDrawingCompleteUPP` (page 120)
: Allocates a Universal Procedure Pointer for the MovieDrawingCompleteProc callback.

`NewMovieExecuteWiredActionsUPP` (page 121)
: Allocates a Universal Procedure Pointer for the MovieExecuteWiredActionsProc callback.

`NewMovieFromDataFork64` (page 123)
: Provides a 64-bit version of NewMovieFromDataFork.

`NewMoviePrePrerollCompleteUPP` (page 132)
: Allocates a Universal Procedure Pointer for the MoviePrePrerollCompleteProc callback.

`NewMoviePreviewCallOutUPP` (page 133)
: Allocates a Universal Procedure Pointer for the MoviePreviewCallOutProc callback.

`NewMovieProgressUPP` (page 134)
: Allocates a Universal Procedure Pointer for the MovieProgressProc callback.

`NewMovieRgnCoverUPP` (page 134)
: Allocates a Universal Procedure Pointer for the MovieRgnCoverProc callback.

`NewMoviesErrorUPP` (page 135)
: Allocates a Universal Procedure Pointer for the MoviesErrorProc callback.

`NewQTCallBackUPP` (page 135)
: Allocates a Universal Procedure Pointer for the QTCallBackProc callback.

2006-05-23 | © 2006 Apple Computer, Inc. All Rights Reserved.

`NewQTEffectListFilterUPP` (page 136)
  Allocates a Universal Procedure Pointer for the QTEffectListFilterProc callback.

`NewQTSyncTaskUPP` (page 137)
  Allocates a Universal Procedure Pointer for the QTSyncTaskProc callback.

`NewTextMediaUPP` (page 141)
  Allocates a Universal Procedure Pointer for the TextMediaProc callback.

`NewTrackTransferUPP` (page 141)
  Allocates a Universal Procedure Pointer for the TrackTransferProc callback.

`NewTweenerDataUPP` (page 142)
  Allocates a Universal Procedure Pointer for the TweenerDataProc callback.

`OpenMovieStorage` (page 145)
  Opens a data handler for movie storage.

`QTDisposeTween` (page 155)
  Disposes of a tween component instance.

`QTDoTween` (page 155)
  Runs a tween component.

`QTGetAtomParent` (page 161)
  Gets the parent of a QT atom.

`QTGetDataHandlerDirectoryDataReference` (page 163)
  Returns a new data reference to the parent directory of the storage location associated with a data handler instance.

`QTGetDataHandlerFullPathCFString` (page 163)
  Returns the full pathname of the storage location associated with a data handler.

`QTGetDataHandlerTargetNameCFString` (page 164)
  Returns the name of the storage location associated with a data handler.

`QTGetDataReferenceDirectoryDataReference` (page 165)
  Returns a new data reference for a parent directory.

`QTGetDataReferenceFullPathCFString` (page 165)
  Returns the full pathname of the target of the data reference as a CFString.

`QTGetDataReferenceTargetNameCFString` (page 166)
  Returns the name of the target of a data reference as a CFString.

`QTGetDataRefMaxFileOffset` (page 167)
  Undocumented

`QTNewAlias` (page 178)
  Creates a Mac OS alias to a file.

`QTNewDataReferenceFromCFURL` (page 179)
  Creates a URL data reference from a CFURL.

`QTNewDataReferenceFromFSRef` (page 180)
  Creates an alias data reference from a file specification.

`QTNewDataReferenceFromFSRefCFString` (page 181)
  Creates an alias data reference from a file reference pointing to a directory and a file name.

`QTNewDataReferenceFromFSSpec` (page 182)
  Creates an alias data reference from a file specification of type FSSpec.

`QTNewDataReferenceFromFullPathCFString` (page 183)
: Creates an alias data reference from a CFString that represents the full pathname of a file.

`QTNewDataReferenceFromURLCFString` (page 184)
: Creates a URL data reference from a CFString that represents a URL string.

`QTNewDataReferenceWithDirectoryCFString` (page 185)
: 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 186)
: Undocumented

`RemoveMovieExecuteWiredActionsProc` (page 198)
: Removes a MovieExecuteWiredActionsProc callback from a movie.

`RemoveSoundDescriptionExtension` (page 199)
: Removes an extension from a SoundDescription structure.

`SetMediaDataRef` (page 201)
: Changes the file that the specified media identifies as the location for its data storage.

`SetMediaDataRefAttributes` (page 202)
: Sets a data reference's attributes.

`SetMovieAnchorDataRef` (page 205)
: Sets a movie's anchor data reference and type.

`SetMovieDefaultDataRef` (page 211)
: Sets a movie's default data reference and type.

`SetMoviePropertyAtom` (page 214)
: Sets a movie's property atom.

`SetMovieVisualBrightness` (page 214)
: Sets the brightness adjustment for the movie.

`SetMovieVisualContrast` (page 215)
: Sets the contrast adjustment for the movie.

`SetMovieVisualHue` (page 216)
: Sets the hue adjustment for the movie.

`SetMovieVisualSaturation` (page 216)
: Sets the color saturation adjustment for the movie.

`SetQuickTimePreference` (page 218)
: Sets a particular preference in the QuickTime preferences.

`SetSpriteWorldFlags` (page 221)
: Sets flags that govern the behavior of a sprite world.

`SetSpriteWorldGraphicsMode` (page 221)
: Sets the graphics transfer mode for a sprite world.

`ShowMovieInformation` (page 226)
: 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` and `GetTrackMedia`. 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 70). 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` and `GetMoviesStickyError`, 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`, `NewMovieFromFile` (page 126), and `NewMovieFromHandle` (page 128).

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` and `GetMoviesStickyError`, 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`, `NewMovieFromFile` (page 126), and `NewMovieFromHandle` (page 128).

resRefNum

Identifies the movie file to which the resource is to be added. Your application obtains this value from the `OpenMovieFile` (page 143) 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` and `GetMoviesStickyError`, 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 29).

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`, `NewMovieFromFile` (page 126), and `NewMovieFromHandle` (page 128).

`dh`

The data handler component that was returned by `CreateMovieStorage` (page 46).

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` and `GetMoviesStickyError`, as well as in the function result. See `Error Codes`.

Discussion

This function calls `PutMovieIntoStorage` internally. If you are writing a custom data handler, make sure it implements `DataHGetDataRef`. Also implement `DataHScheduleData64` and `DataHGetFileSize64`, or `DataHScheduleData` and `DataHGetFileSize` if the data handler does not support 64-bit file offsets, plus `DataHWrite64`, or `DataHWrite` if it does not support 64-bit offsets.

Version Notes

Introduced in QuickTime 6. Supersedes `AddMovieResource` (page 27).

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` and `GetMoviesStickyError`, 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`, `GetTrackUserData`, or `GetMediaUserData`.

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` and `GetMoviesStickyError`, 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`, `GetTrackUserData`, or `GetMediaUserData`.

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` and `GetMoviesStickyError`, 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 63) 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 63) 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` and `GetMoviesStickyError`, 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 63) 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` and `GetMoviesStickyError`, 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` and `GetMoviesStickyError`, 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`, `NewMovieFromFile` (page 126), and `NewMovieFromHandle` (page 128).

Return Value

You can access this function's error returns through `GetMoviesError` and `GetMoviesStickyError`.

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 143).

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` and `GetMoviesStickyError`, 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 40).

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 46).

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` and `GetMoviesStickyError`, as well as in the function result. See `Error Codes`.

Version Notes

Introduced in QuickTime 6. Supersedes `CloseMovieFile` (page 39).

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` and `GetTrackMedia`.

dstMedia

The destination media for this operation. Your application obtains this media identifier from such functions as `NewTrackMedia` and `GetTrackMedia`.

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` and `GetMoviesStickyError`, 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`, `NewMovieFromFile` (page 126), and `NewMovieFromHandle` (page 128).

dstMovie

The destination movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile` (page 126), and `NewMovieFromHandle` (page 128).

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` and `GetMoviesStickyError`, 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` and `GetMovieTrack`.

dstTrack
: The destination track for this operation. Your application obtains this track identifier from such functions as `NewMovieTrack` and `GetMovieTrack`.

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` and `GetMoviesStickyError`, 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`, `GetTrackUserData`, or `GetMediaUserData`.

dstUserData

The destination user data list for this operation. You obtain this list reference by calling `GetMovieUserData`, `GetTrackUserData`, or `GetMediaUserData`.

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` and `GetMoviesStickyError`, 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`, `GetTrackUserData`, or `GetMediaUserData` 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

vrmovies.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` and `GetMoviesStickyError`, 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 46).

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

qteffects

qteffects.win

vrmakepano

VRMakePano Library

2006-05-23 | © 2006 Apple Computer, Inc. All Rights Reserved.

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` and `GetMoviesStickyError`, as well as in the function result. See `Error Codes`.

Discussion
If you are writing a custom data handler, make sure it supports `DataHGetDataRef`. It must also support `DataHWrite64`, or `DataHWrite` if 64-bit offsets are not supported.

Version Notes
Introduced in QuickTime 6. Supersedes `CreateMovieFile` (page 44).

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` and `GetMoviesStickyError`, 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` and `GetMoviesStickyError`, 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 49).

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` and `GetMoviesStickyError`, 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`.

Version Notes

Introduced in QuickTime 6. Supersedes `DeleteMovieFile` (page 48).

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` and `GetMoviesStickyError`.

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` and `GetMoviesStickyError`.

Discussion
This function calls `DisposeSprite` (page 59) 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` and `GetMoviesStickyError`.

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` and `GetMoviesStickyError`.

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 119) function.

Return Value

You can access this function's error returns through `GetMoviesError` and `GetMoviesStickyError`.

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, 0L);  // 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` and `GetMoviesStickyError`.

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` and `GetMoviesStickyError`.

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` and `GetMoviesStickyError`.

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` and `GetMoviesStickyError`.

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` and `GetMoviesStickyError`.

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` and `GetMoviesStickyError`.

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` and `GetMoviesStickyError`.

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` and `GetMoviesStickyError`.

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` and `GetMoviesStickyError`.

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` and `GetMoviesStickyError`.

Discussion

You call this function to dispose of a sprite created by `NewSprite` (page 137). 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` and `GetMoviesStickyError`.

Discussion

You call this function to dispose of a sprite world created by `NewSpriteWorld` (page 139). 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` and `GetMoviesStickyError`.

Version Notes
Introduced in QuickTime 4.1.

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

Related Sample Code
qttext

2006-05-23 | © 2006 Apple Computer, Inc. All Rights Reserved.

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` and `GetMoviesStickyError`.

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` and `GetMoviesStickyError`.

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` and `GetMoviesStickyError`, 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 33).

flags

Reserved. Set this parameter to `NIL`.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` and `GetMoviesStickyError`, as well as in the function result. See `Error Codes`.

2006-05-23 | © 2006 Apple Computer, Inc. All Rights Reserved.

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 33). 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`, `NewMovieFromFile` (page 126), and `NewMovieFromHandle` (page 128).

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` and `GetMoviesStickyError`.

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

2006-05-23 | © 2006 Apple Computer, Inc. All Rights Reserved.

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`, `NewMovieFromFile` (page 126), and `NewMovieFromHandle` (page 128).

`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 64), 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 224) 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`, `NewMovieFromFile` (page 126), and `NewMovieFromHandle` (page 128).

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`, `NewMovieFromFile` (page 126), and `NewMovieFromHandle` (page 128).

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` and `GetMoviesStickyError`, 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 177) 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` and `GetTrackMedia`. 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 71).

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` and `GetMoviesStickyError`, 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` and `GetTrackMedia`. 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` and `GetMoviesStickyError`, 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` and `GetTrackMedia`.

interestingTimeFlags

Flags that determine the search criteria. Note that you may set only one of the `nextTimeMediaSample`, `nextTimeMediaEdit`, or `nextTimeSyncSample` flags to 1. Set unused flags to 0: `nextTimeMediaSample` Set this flag to 1 to search for the next sample. `nextTimeMediaEdit` Set this flag to 1 to search for the next group of samples. `nextTimeSyncSample` Set this flag to 1 to search for the next sync sample. `nextTimeEdgeOK` Set this flag to 1 to accept information about elements that begin or end at the time specified by the `decodeTime` parameter. When this flag is set the function returns valid information about the beginning and end of a media. 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` and `GetTrackMedia`.

interestingTimeFlags

Flags that determine the search criteria. Note that you may set only one of the `nextTimeMediaSample`, `nextTimeMediaEdit`, or `nextTimeSyncSample` flags to 1. Set unused flags to 0: `nextTimeMediaSample` Set this flag to 1 to search for the next sample. `nextTimeMediaEdit` Set this flag to 1 to search for the next group of samples. `nextTimeSyncSample` Set this flag to 1 to search for the next sync sample. `nextTimeEdgeOK` Set this flag to 1 to accept information about elements that begin or end at the time specified by the `decodeTime` parameter. When this flag is set the function returns valid information about the beginning and end of a media. 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` and `GetTrackMedia`. 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` and `GetMoviesStickyError`.

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

2006-05-23 | © 2006 Apple Computer, Inc. All Rights Reserved.

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` and `GetTrackMedia`. 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` and `GetMoviesStickyError`, 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`, `NewMovieFromFile` (page 126), and `NewMovieFromHandle` (page 128).

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` and `GetMoviesStickyError`, 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`, `NewMovieFromProperties`, `NewMovieFromFile`, and `NewMovieFromHandle` (page 128).

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 205).

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`, `NewMovieFromProperties`, `NewMovieFromFile`, and `NewMovieFromHandle` (page 128).

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`, `NewMovieFromProperties`, `NewMovieFromFile`, and `NewMovieFromHandle` (page 128).

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`, `NewMovieFromProperties`, `NewMovieFromFile`, and `NewMovieFromHandle` (page 128).

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 206).

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`, `NewMovieFromProperties`, `NewMovieFromFile`, and `NewMovieFromHandle` (page 128).

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 207).

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`, `NewMovieFromProperties`, `NewMovieFromFile`, and `NewMovieFromHandle` (page 128).

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 207).

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`, `NewMovieFromProperties`, `NewMovieFromFile`, and `NewMovieFromHandle` (page 128).

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`, `NewMovieFromProperties`, `NewMovieFromFile`, and `NewMovieFromHandle` (page 128).

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 208).

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`, `NewMovieFromFile` (page 126), and `NewMovieFromHandle` (page 128).

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` and `GetMoviesStickyError`, 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`, `NewMovieFromFile` (page 126), and `NewMovieFromHandle` (page 128).

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` and `GetMoviesStickyError`, 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`, `NewMovieFromFile` (page 126), and `NewMovieFromHandle` (page 128).

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` and `GetMoviesStickyError`, 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`, `NewMovieFromFile` (page 126), and `NewMovieFromHandle` (page 128).

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`, `NewMovieFromFile` (page 126), and `NewMovieFromHandle` (page 128).

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`, `NewMovieFromFile` (page 126), and `NewMovieFromHandle` (page 128).

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` and `GetMoviesStickyError`.

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`, `NewMovieFromFile` (page 126), and `NewMovieFromHandle` (page 128).

propertyAtom

A pointer to a property atom.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` and `GetMoviesStickyError`, 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`, `NewMovieFromFile` (page 126), and `NewMovieFromHandle` (page 128).

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`, `NewMovieFromFile` (page 126), and `NewMovieFromHandle` (page 128).

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` 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`, `GetTrackUserData`, or `GetMediaUserData`.

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`, `NewMovieFromFile` (page 126), and `NewMovieFromHandle` (page 128).

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` and `GetMoviesStickyError`.

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` and `GetMoviesStickyError`, 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` and `GetMoviesStickyError`, 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` and `GetMoviesStickyError`, 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` and `GetMovieTrack`.

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 223).

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` and `GetMovieTrack`.

muted

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

flags

Not used; set to 0.

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

Discussion

The track's mute setting is not stored in the movie; it is used only until the movie is closed. See `SetTrackAudioMute` (page 223).

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` and `GetMovieTrack`.

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` and `GetMoviesStickyError`.

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` and `GetMovieTrack`.

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` and `GetMoviesStickyError`. 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` and `GetMovieTrack`.

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 91).

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`. 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`, `GetTrackUserData`, or `GetMediaUserData` 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` and `GetMoviesStickyError`, 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`, `GetTrackUserData`, or `GetMediaUserData`.

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` and `GetMoviesStickyError`, 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`, `GetTrackUserData`, or `GetMediaUserData` 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` and `GetMoviesStickyError`, 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`, `NewMovieFromFile` (page 126), and `NewMovieFromHandle` (page 128).

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 38).

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` and `GetMoviesStickyError`. See `Error Codes`.

Discussion

In most cases, you do not need to call this function. When you call `SetSpriteProperty` (page 218) 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` and `GetMoviesStickyError`, 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 229) is called.

2006-05-23 | © 2006 Apple Computer, Inc. All Rights Reserved.

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` and `GetTrackMedia`.

`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` and `GetMoviesStickyError`, 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 108) 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` and `GetMovieTrack`.

`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` and `GetMoviesStickyError`, 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 109) 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`, `NewMovieFromProperties`, `NewMovieFromFile`, and `NewMovieFromHandle` (page 128).

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 112).

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 111).

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 111).

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 111).

`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 111).

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 111).

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`, `NewMovieFromFile` (page 126), and `NewMovieFromHandle` (page 128).

flags

Undocumented See these constants:
`movieExecuteWiredActionDontExecute`

actions

Undocumented

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` and `GetMoviesStickyError`, 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`, `NewMovieFromFile` (page 126), and `NewMovieFromHandle` (page 128).

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` and `GetMoviesStickyError`, 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`

2006-05-23 | © 2006 Apple Computer, Inc. All Rights Reserved.

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` and `GetMoviesStickyError`, 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 128), 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 230) 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` and `GetMoviesStickyError`, 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`.

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 230) 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` and `GetMoviesStickyError`, 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`.

Version Notes

Introduced in QuickTime 4. Superseded in QuickTime 6 by `NewMovieFromStorageOffset` (page 130).

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` and `GetMoviesStickyError`, 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 126). 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`.

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

2006-05-23 | © 2006 Apple Computer, Inc. All Rights Reserved.

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 143) 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 230) to preserve these changes. Set this parameter to `NIL` if you don't want to receive this information. See `NewMovieTrack` 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` and `GetMoviesStickyError`, 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`.

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`.

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` and `GetMoviesStickyError`, 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`.

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`.

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` 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`.

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 46). 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 230) 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` and `GetMoviesStickyError`, as well as in the function result. See `Error Codes`.

Discussion

This function serves the same purpose for data handlers as `NewMovieFromDataFork64` (page 123) 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`. Also implement `DataHScheduleData64` and `DataHGetFileSize64`, or `DataHScheduleData` and `DataHGetFileSize` 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`.

Version Notes

Introduced in QuickTime 6. Supersedes `NewMovieFromDataFork64` (page 123).

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 230) 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` and `GetMoviesStickyError`, 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 126)), 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`.

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` and `GetMoviesStickyError`, 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 218) to assign the image description handle and image data to the sprite before the next call to `SpriteWorldIdle` (page 229).

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` and `GetMoviesStickyError`, 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.

2006-05-23 | © 2006 Apple Computer, Inc. All Rights Reserved.

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 and GetMoviesStickyError, 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 39) 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 27) 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 145).

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` and `GetMoviesStickyError`, as well as in the function result. See `Error Codes`.

Discussion
This function is rarely used. It is an alternative to `OpenMovieFile` (page 143).

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`, `NewMovieFromFile` (page 126), and `NewMovieFromHandle` (page 128).

`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` and `GetMoviesStickyError`, 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` and `GetMoviesStickyError`, 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`, `NewMovieFromFile` (page 126), and `NewMovieFromHandle` (page 128).

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` and `GetMoviesStickyError`, 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` and `GetMoviesStickyError`, 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 154).

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` and `GetMoviesStickyError`, 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 160).

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` and `GetMoviesStickyError`, 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 168) 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 175) and `QTDismissStandardParameterDialog` (page 153).

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` and `GetMoviesStickyError`, 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 175) 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 194).

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` and `GetMoviesStickyError`, 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 151).

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` and `GetMoviesStickyError`, 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` and `GetMoviesStickyError`, 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 178) or `QTCopyAtom` (page 148).

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` and `GetMoviesStickyError`, 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` and `GetMoviesStickyError`, 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` and `GetMoviesStickyError`, 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 155), 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` and `GetMoviesStickyError`, 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 154).

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` and `GetMoviesStickyError`, 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 176) before you call this function. If you call `QTLockContainer`, you should call `QTUnlockContainer` (page 196) 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` and `GetMoviesStickyError`, 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` and `GetMoviesStickyError`, 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 154) 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` and `GetMoviesStickyError`, 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 151).

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` and `GetMoviesStickyError`, 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` and `GetMoviesStickyError`, 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`, `NewMovieFromFile` (page 126), and `NewMovieFromHandle` (page 128).

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` and `GetMoviesStickyError`, 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` and `GetMoviesStickyError`, 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` and `GetMoviesStickyError`, 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` and `GetMoviesStickyError`, 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.

2006-05-23 | © 2006 Apple Computer, Inc. All Rights Reserved.

```
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 151).

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` and `GetMoviesStickyError`, 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 160) to directly access a leaf atom's data. When you have finished accessing a leaf atom's data, you should call `QTUnlockContainer` (page 196). 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`, `NewMovieFromFile` (page 126), and `NewMovieFromHandle` (page 128).

`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` and `GetMoviesStickyError`, 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 69) 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` and `GetMoviesStickyError`, 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` and `GetMoviesStickyError`, 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` and `GetMoviesStickyError`, 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` and `GetMoviesStickyError`, 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` and `GetMoviesStickyError`, 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 189).

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` and `GetMoviesStickyError`, 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` and `GetMoviesStickyError`, 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` and `GetMoviesStickyError`, 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` and `GetMoviesStickyError`, 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` and `GetMoviesStickyError`, 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` and `GetMoviesStickyError`, 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` and `GetMoviesStickyError`, 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 151).

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` and `GetMoviesStickyError`, 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 151) 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` and `GetMoviesStickyError`, 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` and `GetMoviesStickyError`, 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 176) 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` and `GetMoviesStickyError`, 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`, `NewMovieFromFile` (page 126), and `NewMovieFromHandle` (page 128).

proc

A `MovieExecuteWiredActionsProc` callback that was previously installed using `AddMovieExecuteWiredActionsProc` (page 26).

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` and `GetMoviesStickyError`, 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 143).

resId

ID of the resource to be removed.

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` and `GetMoviesStickyError`, 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` and `GetMoviesStickyError`, 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`, `GetTrackUserData`, or `GetMediaUserData`.

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` and `GetMoviesStickyError`, 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`, `GetTrackUserData`, or `GetMediaUserData`.

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` and `GetMoviesStickyError`, 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` and `GetTrackMedia`. 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 70). 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` and `GetMoviesStickyError`, 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` and `GetTrackMedia`. 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 71).

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` and `GetMoviesStickyError`, 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` and `GetTrackMedia`. 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` and `GetMoviesStickyError`. 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.

2006-05-23 | © 2006 Apple Computer, Inc. All Rights Reserved.

```
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` and `GetMoviesStickyError`, 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, NewMovieFromFile (page 126), and NewMovieFromHandle (page 128).

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 and GetMoviesStickyError, 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, NewMovieFromProperties, NewMovieFromFile, and NewMovieFromHandle (page 128).

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 78).

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`, `NewMovieFromProperties`, `NewMovieFromFile`, and `NewMovieFromHandle` (page 128).

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 80).

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`, `NewMovieFromProperties`, `NewMovieFromFile`, and `NewMovieFromHandle` (page 128).

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 81).

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`, `NewMovieFromProperties`, `NewMovieFromFile`, and `NewMovieFromHandle` (page 128).

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`, `NewMovieFromProperties`, `NewMovieFromFile`, and `NewMovieFromHandle` (page 128).

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 83).

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`, `NewMovieFromFile` (page 126), and `NewMovieFromHandle` (page 128).

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` and `GetMoviesStickyError`, 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 84), 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`, `NewMovieFromFile` (page 126), and `NewMovieFromHandle` (page 128).

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` and `GetMoviesStickyError`. 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`, `NewMovieFromFile` (page 126), and `NewMovieFromHandle` (page 128).

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` and `GetMoviesStickyError`, 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`, `NewMovieFromFile` (page 126), and `NewMovieFromHandle` (page 128).

2006-05-23 | © 2006 Apple Computer, Inc. All Rights Reserved.

language

The movie's language or region code; see `Localization Codes`.

Return Value

You can access error returns from this function through `GetMoviesError` and `GetMoviesStickyError`. 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`, `NewMovieFromFile` (page 126), and `NewMovieFromHandle` (page 128).

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` and `GetMoviesStickyError`. 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`, `NewMovieFromFile` (page 126), and `NewMovieFromHandle` (page 128).

`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` and `GetMoviesStickyError`. 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`, `CutMovieSelection`, `CopyMovieSelection`, `AddMovieSelection`, and `InsertMovieSegment`.

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`, `NewMovieFromFile` (page 126), and `NewMovieFromHandle` (page 128).

`propertyAtom`
: A property atom.

Return Value
You can access Movie Toolbox error returns through `GetMoviesError` and `GetMoviesStickyError`, 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`, `NewMovieFromFile` (page 126), and `NewMovieFromHandle` (page 128).

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` and `GetMoviesStickyError`. See `Error Codes`.

Discussion
You define the poster's image by specifying a time in the movie, using `SetMoviePosterTime`. 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` and `GetMoviesStickyError`, 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` and `GetMoviesStickyError`, 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` and `GetMoviesStickyError`, 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` and `GetMoviesStickyError`, 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` and `GetMoviesStickyError`, 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` and `GetMoviesStickyError`, 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` and `GetMovieTrack`.

`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 98).

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` and `GetMovieTrack`.

muted

Pass TRUE to mute the track's audio, FALSE to unmute it.

flags

Not used; set to 0.

Return Value

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

Discussion

The track mute setting is not stored in the movie; it is used only until the movie is closed. See `GetTrackAudioMute` (page 99).

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` and `GetMovieTrack`.

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` and `GetMoviesStickyError`. 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`. 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`. 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`, `GetTrackUserData`, or `GetMediaUserData`.

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` and `GetMoviesStickyError`, 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`, `NewMovieFromFile` (page 126), and `NewMovieFromHandle` (page 128).

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` and `GetMoviesStickyError`. 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` and `GetMoviesStickyError`, 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` and `GetMoviesStickyError`, 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` and `GetMoviesStickyError`, 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`, `NewMovieFromFile` (page 126), and `NewMovieFromHandle` (page 128).

dh

The data handler component that was returned by `CreateMovieStorage` (page 46).

Return Value

You can access Movie Toolbox error returns through `GetMoviesError` and `GetMoviesStickyError`, as well as in the function result. See `Error Codes`.

Discussion

This function, which is similar to `OpenMovieStorage` (page 145), replaces the content of the movie in the storage associated with the specified data handler.

Version Notes

Introduced in QuickTime 6. Supersedes `UpdateMovieResource` (page 230).

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`, `NewMovieFromFile` (page 126), and `NewMovieFromHandle` (page 128).

resRefNum

Identifies the movie file that contains the resource to be changed. Your application obtains this value from `OpenMovieFile` (page 143).

resId

The resource to be changed. This value is obtained from a previous call to NewMovieFromFile (page 126), NewMovieFromDataRef (page 124), or AddMovieResource (page 27). 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 and GetMoviesStickyError, 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 230).

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 131).

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 131) 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`

2006-05-23 | © 2006 Apple Computer, Inc. All Rights Reserved.

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 79)

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 82)

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 147)

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 171)
`QTRestrictionsGetIndClass` (page 190)
`QTRestrictionsGetInfo` (page 191)
`QTRestrictionsGetItem` (page 192)

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 153)
`QTEqualUUIDs` (page 157)

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`

Document Revision History

This table describes the changes to *Movie Toolkit Reference*.

Date	Notes
2006-05-23	New document, based on previously published material, that describes the API for QuickTime Movie Toolkit.

Index

2006-05-23 | © 2006 Apple Computer, Inc. All Rights Reserved.