What's New in QuickTime 6.4 For Mac OS X

QuickTime

2003-09-01

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

Simultaneously published in the United States and Canada.

Contents

What's New in QuickTime 6.4 For Mac OS X

Welcome to QuickTime 6.4 for Mac OS X.

This document provides details of some of the new features, changes, and enhanced capabilities that are available at the API level in QuickTime 6.4. It also includes some code snippets that illustrate how developers can take advantage of these new features and it fully documents the new QuickTime functions, data structures, and callbacks.

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

Documentation and Other Resources

This document is intended to provide QuickTime developers with detailed information to support their programming and development efforts. It is designed to supplement the information provided in *Inside QuickTime: API Reference* and the suite of QuickTime documentation available online, in HTML and PDF, at http://developer.apple.com/documentation/QuickTime/QuickTime.html.

Updates to the QuickTime technical documentation website are provided on a regular basis. Developers can also subscribe to various mailing lists for the latest news and information. To sign up for any of Apple's Developer Programs, go to http://developer.apple.com/membership/index.html.

Installing QuickTime 6.4

QuickTime 6.4 is available as a standalone download for Mac OS X version 10.2.5 and later. It is also included as a part of Mac OS X version 10.3 The download site is `www.apple.com/quicktime/download/`.

Macintosh users can also use the Software Update mechanism in Mac OS X to update QuickTime 6.0, 6.0.1, 6.0.2, 6.1, 6.1.1, 6.2, or 6.3 to QuickTime 6.4.

Hardware Requirements

QuickTime 6.4 requires the following minimum hardware configuration:

- Mac OS X version 10.2.5 or later
- PowerPC G3 or better running at 400 MHz or higher
- At least 128 MB of RAM

Updating Earlier Versions

QuickTime 6.4 replaces and updates various point releases of QuickTime 6 for Mac OS X.

Important: A Mac OS 9 version is not included in QuickTime 6.4. QuickTime 6.0.3 was the last Mac OS 9 version available to QuickTime users.

The QuickTime 6.4 system software, including the QuickTime Player application, is a free upgrade for QuickTime 6 users. No new Pro key is required; QuickTime 6 Pro keys will unlock the Pro features of QuickTime 6.4.

Overview

The QuickTime API comprises more than 2500 functions that provide services to applications. These services include audio and video capture and playback, movie editing, composition, and streaming, still image display, audio-visual interactivity, and so on. The API also supports a wide range of standards-based formats. It is dedicated to extending the reach of application developers by letting them invoke the full range of QuickTime capabilities.

The QuickTime API is not static, however. It evolves to adopt new idioms, new data structures, and new ways of doing things––all of which continually make the API more convenient for developers to use in their applications.

This document is written for developers who use QuickTime on the Mac OS X platform and who want to learn about new ways of programming with QuickTime 6.4.

QuickTime 6 In Perspective

QuickTime 6, introduced in 2002, represented a major advance in Apple technology. Because QuickTime 6 supports ISO-compliant MPEG-4 video and audio, both encode and decode, you can create and play back MPEG-4 video and audio content and use Advanced Audio Coding (AAC).

In subsequent releases of QuickTime 6, additional support was provided for 3GPP authoring, playback, and delivery. These releases also built on Apple's support of MPEG-4 as the standard for digital media streaming on the Internet and extended it with support for standards for mobile Internet streaming.

New Features of QuickTime 6.4

QuickTime 6.4 now adds significant new functionality to QuickTime. The following new features are discussed in the rest of this document:

- "A New Approach To Data References" (page 8) describes how you can now manipulate QuickTime media via data references—opening and flattening movies, exporting graphics images, and so on.
- "Threaded Programming and QuickTime" (page 12) tells how QuickTime 6.4 supports execution of background tasks on multiple threads in a preemptive multitasking environment. This new technology makes it possible to offload many tasks from your program's main thread.

- "New Graphics Functions" (page 16) explains how you can now use QuickTime to read and write image files while using Core Graphics to draw and manage images.
- "New Graphics Importer Support for ColorSync" (page 19) introduces new graphics importer functions that provide support for ColorSync on Mac OS X.
- "New AV Startup Synchronization Functions" (page 19) describes new functions that improve audio-visual startup synchronization.
- "Processing Events" (page 20) gives some guidelines for handling QuickTime events on the current Mac OS X platform.
- "New Component Properties Functions" (page 21) provides details of QuickTime's new Component Properties API, which let you configure QuickTime processes such as export and recompression without using the QuickTime user interface.
- "New Movie Property Functions" (page 23) describes five movie property functions that are similar in purpose to the component functions described in the previous section.
- "New IIDC Digitizer Functions" (page 23) introduces new APIs to communicate with video digitizers that have a subtype of `vdSubtypeIIDC`. These digitizers support new IEEE-1394-based digital cameras and webcams that have Instrumentation and Industrial Control (IIDC) features.
- "New Sound Function" (page 27) describes a new sound function that identifies the audio device used by a video output component.
- "New Offset TimeBase Functions" (page 27) lists two offset timebase functions that help custom media handlers implement media latency.
- "Changes to Text Drawing" (page 27) explains how QuickTime now uses the ATSUI text drawing engine instead of TextEdit.
- "Encoding Text Changes" (page 28) describes changes to QuickTime's internal behavior for encoding text.
- "New Release of QuickTime for Java" (page 28) discusses QuickTime for Java 1.4.1, which provides new Java functionality for both Mac OS X and Windows.

"QuickTime 6.4 API Reference" (page 33) provides interface details for all the functions, data structures, and callbacks that are new in QuickTime 6.4.

Summary of QuickTime 6 Versions

The following table summarizes the different point releases of QuickTime 6.

QuickTime version	Mac OS X	Windows	Mac OS 9	Features
6	x	x	x	MPEG-4 and lots more.
6.01	x	x	x	Bug fix for QuickTime 6. Last version for all three platforms.
6.03			x	Bug fixes to address security issues. Mac OS 9 only.
6.1	x	x		Improved MPEG-4 video, full-screen modes, wired actions.

QuickTime version	Mac OS X	Windows	Mac OS 9	Features
6.2	x			Support for iTunes 4, enhanced AAC codec, limited DRM.
6.3	x	x		Improved AAC codec, 3GPP support, which includes AMR codec.
6.4 for Mac OS X	x			New data reference functions, true multithreading, new graphics functions, component and movie property access, other API additions.

Using Gestalt to Get the QuickTime Version

As always, the standard way for Apple developers to determine which version of QuickTime is installed is by calling the Macintosh Toolbox `Gestalt` function.

The following code snippet demonstrates how you can check the version of QuickTime that is installed––in this case, QuickTime 6.4. The number `0x06408000` tests for the shipping version of QuickTime 6.4 but fails on prerelease versions.

```
/* check the version of QuickTime installed */
long     version;
OSErr    result;
result = Gestalt(gestaltQuickTime, &version);
if ((result == noErr) && (version >= 0x06408000))
{
    /* we have version 6.4! */
}
```

A New Approach To Data References

The abstraction of a data reference is central to manipulating media in QuickTime. In the past, QuickTime relied on a specialized set of functions for dealing with files and another specialized set of functions that were less rich for dealing with data references. In QuickTime 6.4, this has changed.

With QuickTime 6.4, you can do anything to QuickTime media via a data reference. For example, you can use data references to open movies, flatten movie files, export graphics images, and so on. As file systems change, and as developers invent new ways to refer to media, the QuickTime API offers an abstraction that will grow to meet new needs.

The new data reference functions let you create data references from different forms of file specifications, such as full paths, URLs, and even `FSSpec` structures. This process is illustrated in "Data Reference Example Code" (page 10).

QuickTime 6.4 also provides a set of functions that allow you to import, export, create, and flatten media files that are specified by data reference. The following code samples illustrate the use of data references for these operations:

- Opening a new movie from media specified by data ref, using the existing QuickTime function `NewMovieFromDataRef` (page 13) (page 10).
- Instantiating a graphics importer for an image specified by a data reference, using `GetGraphicsImporterForDataRefWithFlags` (page 21) (page 16).
- Exporting a movie to a location specified by a data reference, using `ConvertMovieToDataRef` (page 13) (page 10).
- Flattening media data to a location specified by a data reference, using `FlattenMovieDataToDataRef` (page 13) (page 10).
- Exporting a graphic image to a location specified by a data reference, using `GraphicsExportSetOutputDataReference` (page 21) (page 16).

You can create a data reference for practically any data format and location, using the functions listed in the next section.

New Data Reference Functions

A number of data reference utility functions are new in QuickTime 6.4.

Seven new functions create new data references from various file specifications, pathnames, and URLs:

- `QTNewDataReferenceFromFSRef` (page 72) creates an alias data reference from a file specification.
- `QTNewDataReferenceFromFSRefCFString` (page 72) creates an alias data reference from a file reference pointing to a directory and a file name.
- `QTNewDataReferenceFromFSSpec` (page 73) creates an alias data reference from a file specification of type `FSSpec`.
- `QTNewDataReferenceWithDirectoryCFString` (page 76) creates an alias data reference from another alias data reference pointing to the parent directory and a `CFString` that contains the file name.
- `QTNewDataReferenceFromFullPathCFString` (page 74) creates an alias data reference from a `CFString` that represents the full pathname of a file.
- `QTNewDataReferenceFromCFURL` (page 71) creates a URL data reference from a CFURL.
- `QTNewDataReferenceFromURLCFString` (page 75) creates a URL data reference from a `CFString` that represents a URL string.

Three functions return information about data references:

- `QTGetDataReferenceDirectoryDataReference` (page 67) returns a new data reference for a parent directory.
- `QTGetDataReferenceTargetNameCFString` (page 69) returns the name of the target of a data reference as a `CFString`.
- `QTGetDataReferenceFullPathCFString` (page 68) returns the full pathname of the target of the data reference as a `CFString`.

Three functions return information about the storage location associated with a data handler:

- QTGetDataHandlerDirectoryDataReference (page 65) returns a new data reference to the parent directory of the storage location associated with a data handler instance.
- QTGetDataHandlerTargetNameCFString (page 66) returns the name of the storage location associated with a data handler.
- QTGetDataHandlerFullPathCFString (page 66) returns the full pathname of the storage location associated with a data handler.

Six functions support data references for imported graphic images:

- GraphicsImportDoExportImageFileToDataRefDialog (page 47) presents a dialog box that lets the user save an imported image in a foreign file format.
- GraphicsImportExportImageFileToDataRef (page 48) saves an imported image in a foreign file format.
- GraphicsImportSaveAsPictureToDataRef (page 51) creates a storage location that contains a QuickDraw picture for an imported image.
- GraphicsImportSaveAsQuickTimeImageFileToDataRef (page 51) creates a storage location that contains a QuickTime image of an imported image.
- MovieImportDoUserDialogDataRef (page 53) requests that a movie import component display its user dialog box.
- MovieImportSetMediaDataRef (page 54) specifies a storage location that is to receive imported movie data.

Two functions perform data reference conversions on movies:

- ConvertMovieToDataRef (page 37) converts a specified movie (or a single track within a movie) into a specified file format and stores it in a specified storage location.
- ConvertDataRefToMovieDataRef (page 35) converts a piece of data in a storage location to a movie file format and stores it in another storage location, supporting a user settings dialog box for import operations.

Data Reference Example Code

The following code snippets show how you can read and write movies using various data references derived from file paths and URLs.

```
Movie movieFromPath(CFStringRef path, BOOL allowQTUserInteraction)
{
    Movie     qtMovie = NULL;
    Handle    dataRef = NULL;
    OSType    dataRefType;
    OSErr     err;

    err = QTNewDataReferenceFromFullPathCFString(path,
                    kQTNativeDefaultPathStyle, 0, &dataRef, &dataRefType);

    if (NULL != dataRef) {
        err = NewMovieFromDataRef( &qtMovie,
                     ( allowQTUserInteraction ? 0 :
                                 newMovieDontAskUnresolvedDataRefs ),
```

```
                        NULL, dataRef, dataRefType );
        DisposeHandle(dataRef);
    }
    return qtMovie;
}

// From Cocoa URLs

Movie movieFromURL(NSURL *url, BOOL allowQTUserInteraction)
{
    Movie     qtMovie = NULL;
    Handle    dataRef = NULL;
    OSType    dataRefType;
    OSErr     err;

    err = QTNewDataReferenceFromCFURL((CFURLRef)url, 0, &dataRef,
                                      &dataRefType);
    if (NULL != dataRef) {
        err = NewMovieFromDataRef( &qtMovie,
                    ( allowQTUserInteraction ? 0 :
                              newMovieDontAskUnresolvedDataRefs ),
                    NULL, dataRef, dataRefType );
        DisposeHandle(dataRef);
    }
    return qtMovie;
}
// Writing to movie files
// Save with dependencies to a file

void writeMovieToFile(Movie qtMovie, CFStringRef path)
{
    Handle         dataRef = NULL;
    OSType         dataRefType;
    DataHandler    dataHandler;

    err = QTNewDataReferenceFromFullPathCFString(path,
                kQTNativeDefaultPathStyle, 0, &dataRef, &dataRefType);
    err = CreateMovieStorage(dataRef, dataRefType, kHFCarbonCreatorCode,
                kScriptTag,
    createMovieFileDeleteCurFile, &dataHandler, NULL);
    AddMovieToStorage(qtMovie, dataHandler);
    CloseComponent(dataHandler);
    DisposeHandle(dataRef);
}

// Cocoa variant of the above using NSData

- (void) writeMovie:(Movie)qtMovie toFile:(NSString *)path
{
    Handle    publicMovieHndl;
    NSData    *movieData;

    publicMovieHndl = NewHandle(0);
    PutMovieIntoHandle(qtMovie, publicMovieHndl);
    HLock(publicMovieHndl);
    movieData = [NSData dataWithBytes:*publicMovieHndl
                      length:GetHandleSize(publicMovieHndl)];
    DisposeHandle(publicMovieHndl);
```

```
    [movieData writeToFile:path atomically:YES];

    // set file attributes via NSFileManager
}



void writeFlattenedMovieToFile(Movie qtMovie, CFStringRef path)
{
    Handle    dataRef = NULL;
    OSType    dataRefType;

    err = QTNewDataReferenceFromFullPathCFString(path,
             kQTNativeDefaultPathStyle, 0, &dataRef, &dataRefType);
    err = FlattenMovieDataToDataRef(qtMovie, flattenAddMovieToDataFork,
                             dataRef, dataRefType, kHFSCreatorCode,
                             smSystemScript, createMovieFileDeleteCurFile);
    DisposeHandle(dataRef);
}
```

The following code illustrates how you can export a movie to a WAV file.

```
- (void) writeSoundMovieAsWAVEFile (Movie theMovie, NSString *path)
 {
    Handle         dataRef = NULL;
    OSType         dataRefType;
    DataHandler    dataHandler;

    err = QTNewDataReferenceFromFullPathCFString((CFStringRef)path,
              kQTNativeDefaultPathStyle, 0, &dataRef, &dataRefType);

    // use the default progress procedure, if any
    SetMovieProgressProc(theMovie, (MovieProgressUPP)-1L, 0);
    // export the movie into a file
    ConvertMovieToDataRef( theMovie,        // the movie to convert
                           NIL,             // all tracks in the movie
                           dataRef,         // the output data reference
                           dataRefType,     // the data ref type
                           kQTFileTypeWave, // the output file type
                           kHFSCreatorCode, // the output file creator
                           0L,              // no flags
                           NIL);            // no specific component

    DisposeHandle(dataRef);
}
```

Threaded Programming and QuickTime

QuickTime 6.4 introduces several new features that support execution of background tasks on multiple threads in a preemptive multitasking environment. This makes it possible to offload many tasks from your program's main thread to forestall blocking the user interface. Typical thread-safe tasks include importing still images or image sequences, exporting movies, and rendering to the offscreen graphics world a movie that is not currently being played.

The new threading features allow a degree of concurrent tasking in QuickTime that was not previously possible. In the past, QuickTime could be used on separate threads only by serializing access to the QuickTime API, either by the application itself or through the use of a cooperative threading manager such as the Carbon Thread Manager. While serializing allowed QuickTime to be used safely on multiple threads, it did not support concurrent QuickTime operations; if one thread was executing a QuickTime function, other threads needing to call QuickTime were blocked.

Getting Ready to Use QuickTime from a Thread

To use QuickTime in the background on a preemptive thread, make a call to `EnterMoviesOnThread` (page 39) from the thread before calling any other QuickTime functions from that thread. It is important that you call `EnterMovies` on your main thread before spawning any threads that call `EnterMoviesOnThread`.

`EnterMoviesOnThread` initializes QuickTime with an environment that is private to the thread. Calls to `GetMoviesError` or `MoviesTask(null)`, for example, will not obtain errors or task movies in other threads.

User Interface Limited to the Main Thread

Currently, only the main thread can access the user interface. You can play movies or open user dialogs only from the main thread.

To perform an export operation that requires a user dialog, for example, you need to first execute the dialog on the main thread. You can then perform the actual export operation on a separate thread without tying up the user interface. You can pass the information returned by the dialog to another thread by passing an atom container, using functions that get settings as atom containers and set settings from atom containers.

Error Handling

Calling `EnterMoviesOnThread` indicates that QuickTime should perform additional thread-safety checks on components opened and operations performed on the thread. If you call a function that requires use of a non-thread-safe component, or requires access to the user interface, or performs another thread-unsafe operation, QuickTime returns a distinguished error (`componentNotThreadSafeErr` = -2098).

Not all QuickTime components are thread-safe, so your code should be designed to detect threading error messages and transfer necessary tasks to the main thread. For example, you might spawn a thread to import a list of image files. The thread would import all the image files that have thread-safe importer components and return a list of any unhandled cases to the main thread, which could then import any remaining image files in the foreground.

From the main thread, your application can call `CSSetComponentsThreadMode`, passing it `kCSAcceptThreadSafeComponentsOnlyMode`, to see if opening a movie or other QuickTime object will succeed when attempted from a preemptive thread. You can do this only with Mac OS X version 10.3 or later.

Using QuickTime From a Thread

It is important that you use multiple threads to perform tasks on different movies. Do not use multiple threads to act on the same movie concurrently.

You can work on a given movie in separate threads sequentially, for example to perform an export in the background after playing a movie in the foreground, by passing a data reference from one thread to another when you are through operating on the movie in the first thread.

QuickTime 6.4 includes the following functions that you can use to associate movies and time bases with threads:

- `AttachMovieToCurrentThread` (page 33) attaches a movie to the current thread.
- `AttachTimeBaseToCurrentThread` (page 33) attaches a time base to the current thread.
- `DetachMovieFromCurrentThread` (page 39) detaches a movie from the current thread.
- `DetachTimeBaseFromCurrentThread` (page 39) detaches a time base from the current thread.
- `GetMovieThreadAttachState` (page 42) determines whether a given movie is attached to a thread.
- `GetTimeBaseThreadAttachState` (page 44) determines whether a given time base is attached to a thread.

You can open movies in separate threads from the same source file, but each thread creates its own movie from the file. You can, however, play one movie while exporting the other, for example, which allows your application to behave as if it could concurrently process the same movie on different threads. This currently works with movies whose data is accessed from files, but not with movies accessed from a URL. The URL data handler is not currently thread-safe, so it is not possible to work with separate movies from the same URL in different threads.

Similarly, a thread-safe component type can be used by multiple threads at the same time, but each thread must instantiate its own instance of the component. Do not call a single component instance from multiple threads.

Many QuickTime functions that expect a component instance as a parameter also accept a component in that parameter. This ambiguity should generally be avoided when using different instances of the same component from multiple threads.

Cleaning Up

When your thread is done working with QuickTime, call `ExitMoviesOnThread` (page 40) prior to closing the thread. Failure to do so may cause a memory leak, because resources allocated by `EnterMoviesOnThread` for the private QuickTime environment may fail to be released.

You may call `EnterMoviesOnThread` multiple times, which allows libraries to use this function without needing to know if their host thread has already done so. Subsequent calls do little more than increment a counter. To prevent memory leaks, one call to `ExitMoviesOnThread` should be made for each call to `EnterMoviesOnThread`.

Never call `ExitMoviesOnThread` without a prior call to `EnterMoviesOnThread`. Calls may be nested, but each instance of `ExitMoviesOnThread` must be balanced by a prior call to `EnterMoviesOnThread`.

Backward Compatibility

Because QuickTime did not previously support concurrent use from multiple threads, programs that already use QuickTime in preemptive threads may, accidentally or intentionally, make use of formerly global QuickTime states and data structures. For example, in older versions of QuickTime a call to `MoviesTask(null)` gives processor time to all movies in any thread. Similarly, an error handling routine in one thread might successfully detect errors in another.

While this kind of cross-thread interaction is more likely to do harm than good, the possibility exists that existing applications may rely on it. Consequently, if a thread makes calls to QuickTime without calling `EnterMoviesOnThread`, the thread shares QuickTime's state and data structures on the main thread and any other threads that have not called `EnterMoviesOnThread`.

It is strongly recomended that you transition existing code away from any dependance on QuickTime states and data structures across threads as quickly as possible. When writing new code, threads that call QuickTime should use `EnterMoviesOnThread` to create private, thread-specific versions of the QuickTime environment.

Thread Safety Issues

For developers who are not familiar with QuickTime's existing API, it's possible to assume––largely because Mac OS X is multithreaded––that QuickTime is thread-safe. This is not the case. Your application can't call QuickTime from arbitrary threads.

However, in some cases and with great care, in QuickTime 6.4 you are able to perform a few operations on secondary threads. The key point is that you can't do it willy-nilly. If you need to use QuickTime on secondary threads, in limited cases, it is now possible.

The rules for new threads that call QuickTime are as follows:

- Call `CSSetComponentsThreadMode(kCSAcceptThreadSafeComponentsOnlyMode)` early on. This will instruct the Component Manager not to open non-thread-safe components from this thread.
- If you get `componentNotThreadSafeErr` from a QuickTime API call, the main thread must do the work instead.

Component developers should make their components thread-safe and set the new component flag `cmThreadSafe`.

The following parts of Mac OS X are newly thread-safe:

- QuickDraw
- the Component Manager
- the Alias Manager
- the Memory Manager (`MemError` is now per-thread.)

The following are not thread-safe:

- the Resource Manager
- Component `RefCons` for shared globals

In summary, you should unblock your user interface by moving slow QuickTime processing to other threads, cope with dynamic discovery of non-thread-safe media, and make your components thread-safe.

New Graphics Functions

QuickTime's graphics import components now have the ability to provide Mac OS X Core Graphics `CGImage` images. As a result, you can now specify a `CGImage` as the source for an image export operation. With QuickTime 6.4, you can combine the use of QuickTime to read and write image files with the use of Core Graphics to draw or manage images.

QuickTime 6.4 includes five new functions for working with Core Graphics:

- `GraphicsImportCreateCGImage` (page 46) imports an image as a Core Graphics `CGImage`.
- `GraphicsExportSetInputCGImage` (page 46) specifies a Core Graphics `CGImage` as the source for a graphics export operation.
- `GraphicsExportGetInputCGImage` (page 45) determines which Core Graphics `CGImage` is the source for a graphics export operation.
- `GraphicsExportSetInputCGBitmapContext` (page 45) sets the `CGBitmapContext` that the graphics exporter will use as its input image.
- `GraphicsExportGetInputCGBitmapContext` (page 44) retrieves the `CGBitmapContext` that the graphics exporter is using as its input image.

The following sample code illustrates how to use existing graphics importer functions with the new Core Graphics functions and data reference utilities:

```
// Open a still image from a file
OpenGraphicsImportComponentimageFromPath(CFStringRef path)
{
    ComponentResult           result;
    GraphicsImportComponent   grip = NULL;

    result = QTNewDataReferenceFromFullPathCFString(path,
                    kQTNativeDefaultPathStyle, 0, &dataRef, &dataRefType);
    if (NULL != dataRef) {
        GetGraphicsImporterForDataRefWithFlags(dataRef, dataRefType,
                                               &grip, 0);
        DisposeHandle(dataRef);
    }
    return grip;
}

// Export a still image to a file

OSStatus exportImageToPNGFile(GraphicsImportComponent imageGrip,
                              CFStringRef path)
{
    Handle                    dataRef = NULL;
    OSType                    dataRefType;
    GraphicsExportComponent   graphicsExporter;
    unsigned long             sizeWritten;
    ComponentResult           result;
```

```
    result = QTNewDataReferenceFromFullPathCFString(path,
                    kQTNativeDefaultPathStyle, 0, &dataRef, &dataRefType);
    result = OpenADefaultComponent(GraphicsExporterComponentType,
                                    kQTFileTypePNG, &graphicsExporter);
    result = GraphicsExportSetInputGraphicsImporter(graphicsExporter,
                                                    imageGrip);
    result = GraphicsExportSetOutputDataReference(graphicsExporter,
                                                  dataRef, dataRefType);
    result = GraphicsExportDoExport(graphicsExporter, &sizeWritten);
    CloseComponent(graphicsExporter);
    DisposeHandle(dataRef);
    return result;
}
// Drawing movies and images in windows and offscreen buffers

void playMovieToWindow(Movie qtMovie, Rect *movieBounds, WindowRef window)
{
    Rect        qdRect;
    CGrafPtr    windowPort;
    OSErr       err;

    SetMovieBox(qtMovie, movieBounds);
    windowPort = GetWindowPort(window);
    SetMovieGWorld(qtMovie, windowPort, NULL);

    // set the movie's rate to start it playing
}

void setMovieToRenderToOffscreenBuffer(Movie qtMovie, unsigned long
         pixelFormat,CGSize outputSize, void *buffer, size_t bytesPerRow)
{
    Rect          qdRect;
    GWorldPtr     gWorld = NULL;
    OSErr         err;

    SetRect(&qdRect, 0, 0, outputSize.width, outputSize.height);
    err = NewGWorldFromPtr(&gWorld, pixelFormat, &qdRect, NULL, NULL, 0,
                           buffer, bytesPerRow);
    SetMovieGWorld(qtMovie, gWorld, NULL);

    // set the movie's rate to start it playing
}

void drawImageToOffscreenBuffer(GraphicsImportComponent grip, unsigned long
          pixelFormat, CGSize outputSize, void *buffer, size_t bytesPerRow)
{
    Rect         qdRect;
    GWorldPtr    gWorld = NULL;
    OSErr        err;

    SetRect(&qdRect, 0, 0, outputSize.width, outputSize.height);
    err = NewGWorldFromPtr(&gWorld, pixelFormat, &qdRect, NULL, NULL, 0,
                           buffer, bytesPerRow);
    GraphicsImportSetGWorld(grip, gWorld, NULL);
    GraphicsImportDraw(grip);
}
```

```
OSStatus drawMovieImageToCGContext(Movie qtMovie, CGContextRef ctx,
                                                        CGRect drawRect)
{
    TimeValue             movieTime;
    PicHandle             picHndl;
    CGDataProviderRef     dataProvider;
    QDPictRef             pictDataRef;
    OSStatus              result;

    movieTime = GetMovieTime(qtMovie, NULL);
    picHndl = GetMoviePict(qtMovie, movieTime);
    HLock((Handle)picHndl);
    dataProvider = CGDataProviderCreateWithData ( NULL, *picHndl,
                                GetHandleSize((Handle)picHndl), NULL );
    pictDataRef = QDPictCreateWithProvider(dataProvider);
    result = QDPictDrawToCGContext(ctx, drawRect, pictDataRef);
    QDPictRelease(pictDataRef);
    CGDataProviderRelease(dataProvider);
    KillPicture(picHndl);
    return result;
}

// Combining Graphics importers/exporters with Core Graphics

CGImageRef getCGImageFromPath(CFStringRef path)
{
    CGImageRef                  imageRef = NULL;
    Handle                      dataRef = NULL;
    OSType                      dataRefType;
    GraphicsImportComponent     gi;
    ComponentResult             result;

    result = QTNewDataReferenceFromFullPathCFString(path,
                    kQTNativeDefaultPathStyle, 0, &dataRef, &dataRefType);
    if (NULL != dataRef) {
       GetGraphicsImporterForDataRefWithFlags(dataRef, dataRefType, &gi, 0);
       result = GraphicsImportCreateCGImage(gi, &imageRef, 0);
       DisposeHandle(dataRef);
       CloseComponent(gi);
    }
    return CGImageRef;
}

OSStatus exportCGImageToPNGFile(CGImageRef imageRef, CFStringRef path)
{
    Handle                    dataRef = NULL;
    OSType                    dataRefType;
    GraphicsExportComponent   graphicsExporter;
    unsigned long             sizeWritten;
    ComponentResult           result;

    result = QTNewDataReferenceFromFullPathCFString(path,
                   kQTNativeDefaultPathStyle, 0, &dataRef, &dataRefType);
    result = OpenADefaultComponent(GraphicsExporterComponentType,
                   kQTFileTypePNG, &graphicsExporter);
    result = GraphicsExportSetInputCGImage(graphicsExporter, imageRef);
    result = GraphicsExportSetOutputDataReference(graphicsExporter,
                                                  dataRef, dataRefType);
```

```
    result = GraphicsExportDoExport(graphicsExporter, &sizeWritten);
    CloseComponent(graphicsExporter);
    DisposeHandle(dataRef);
    return result;
}
```

New Graphics Importer Support for ColorSync

QuickTime 6.4 introduces new graphics importer functions that provide support for ColorSync. These six functions are implemented only in Mac OS X:

- `GraphicsImportWillUseColorMatching` (page 53) asks whether the QuickTime function `GraphicsImportDraw` will use color matching if called with the current importer settings.
- `GraphicsImportGetGenericColorSyncProfile` (page 49) retrieves the generic colorsync profile for a graphics importer component.
- `GraphicsImportSetDestinationColorSyncProfileRef` (page 52) sets the ColorSync profile for a graphics importer component.
- `GraphicsImportGetDestinationColorSyncProfileRef` (page 49) retrieves a ColorSync profile from a graphics importer component.
- `GraphicsImportSetOverrideSourceColorSyncProfileRef` (page 52) sets the override ColorSync profile for a graphics importer component.
- `GraphicsImportGetOverrideSourceColorSyncProfileRef` (page 50) retrieves the override ColorSync profile for a graphics importer component.

The functions make ColorSync matching the default behavior for naive applications that use graphics importers, that is, for image files with embedded ColorSync profiles and for image files with CMYK image data, with or without embedded ColorSync profiles.

An opt-out flag is provided for applications that call ColorSync directly. `GraphicsImportGetGenericColorSyncProfile` (page 49) substitutes a generic profile in place of an embedded profile when the opt-out flag is not set.

Note that this is a format-specific importer change; third-party importers must revise their applications in order to get this behavior. Applications can override the embedded image profile and can set the destination profile to be matched. If no destination profile is set, a generic profile is used.

The Photoshop graphics importer and graphics exporter now support embedded ColorSync profiles. ColorSync matching also works for CMYK and 16-bit-per-channel QTIF files.

New AV Startup Synchronization Functions

QuickTime 6.4 introduces five new functions and one new flag that deal with audio-visual startup synchronization issues.

The advantage of these new functions is that now QuickTime can be sample-accurate when starting a movie that contains both a video and an audio track. This means that hardware implementations can specify an edge to start a time base, if they provide their own clock, which could represent their `vSync`. This approach can be used for recording and eliminating the long first frame duration that occurs in the current QuickTime implementation.

Five functions new in QuickTime 6.4 are the following:

- `ClockGetTimeForRateChange` (page 34) obtains the current time according to a specific clock, preferred time, and safe increment duration.
- `ClockGetRateChangeConstraints` (page 34) lets you obtains a minimum delay and maximum delay that a clock could introduce during rate change.
- `GetTimeBaseRateChangeStatus` (page 43) lets a time base client find out about the last rate change status of its time base.
- `TimeBaseStatus` flags. New flags are returned by `GetTimeBaseRateChangeStatus` (page 43) when the clock is waiting for a future time to start moving while its rate is nonzero.
- `GetMovieRateChangeConstraints` (page 41) returns the minimum and maximum delay you can get when the movie rate changes.
- `ConvertTimeToClockTime` (page 38) converts a time record expressed in a time base to the clock time. This function was added in an earlier release of QuickTime and is now public, since `GetTimeBaseRateChangeStatus`, which is new, returns parameters expressed in clock time.

How Startup Synchronization Works

A time base may run at a nonzero rate, but the time will not move until a specific clock value is reached in order to make sure that each track can see the start time of the movie in the future.

When the rate changes to nonzero, the time base synchronizes its time with the current clock time by calculating an offset. It then tells the clock what it has done in order to reschedule any pending callbacks. An extra delay is added in this offset so that the time base time can be frozen to the current time until the delay is reached. The clock will ask the time base about this offset to properly adjust any `atTime` callback until the offset is reached.

The synchronization point between the time base and a clock is accomplished using the new functions listed in the previous section. Hardware developers should be able to take advantage of them in their applications.

Processing Events

The current and future evolution of the Mac OS is shifting some of the ways that events are handled by QuickTime. Here are some advisories:

- QuickTime's movie controller function `MCIsPlayerEvent` requires user events such as mouse clicks and keypresses to be passed to movies via the classic Event Manager `EventRecord`. However, with the advent of Carbon events, the `EventRecord` is no longer in common use. Use `MCClick` and `MCKey` instead.

- `MCIsPlayerEvent` was able to take advantage of classic null events in order to provide movies with sufficient processing time to render video frames and to perform other tasks. However, null events are delivered only to applications that implement classic event loops which call `WaitNextEvent`; modern applications using Carbon or Cocoa do not implement event loops in the classic fashion. Use `MCIdle` instead.
- Instead of implementing the details of passing user events and of idling movies yourself, Apple recommends the use of higher level modules that take care of the details for you. For Carbon applications, use the Carbon Movie Control. For Cocoa applications, use `NSMovieView`. Both of these modules provide support for displaying and playing back the complete range of QuickTime media and also support copy-and-paste editing.

New Component Properties Functions

QuickTime-based applications increasingly require that media capture, import, export, and compression operations be configured from custom user interfaces or configured automatically, without user intervention.

One of the changes in QuickTime 6.4 is the ability to use a new Component Properties API so that you can configure QuickTime processes such as export and recompression without using the QuickTime user interface. You can then customize the user interface to fit your application's needs.

The following eleven new functions in QuickTime 6.4 help you work with component properties:

- `QTGetComponentPropertyInfo` (page 64) returns information about the properties of a component.
- `QTGetComponentProperty` (page 63) returns the value of a specific component property.
- `QTSetComponentProperty` (page 78) sets the value of a specific component property.
- `QTAddComponentPropertyListener` (page 55) installs a callback to monitor a component property.
- `QTRemoveComponentPropertyListener` (page 76) removes a component property monitoring callback.
- `QTComponentPropertyListenerCollectionCreate` (page 58) creates a collection of component property monitors.
- `QTComponentPropertyListenerCollectionAddListener` (page 56) adds a listener callback for a specified property class and ID to a property listener collection.
- `QTComponentPropertyListenerCollectionRemoveListener` (page 62) removes a listener callback with a specified property class and ID from a property listener collection.
- `QTComponentPropertyListenerCollectionNotifyListeners` (page 60) calls all listener callbacks in a component property listener collection registered for a specified property class and ID.
- `QTComponentPropertyListenerCollectionHasListenersForProperty` (page 58) determines if there are any listeners in a component property listener collection registered for a specified property class and ID.
- `QTComponentPropertyListenerCollectionIsEmpty` (page 59) determines if a listener collection is empty.

To date, the Component Manager has defined all component selectors that are standard across multiple types of components, whether these selectors are required or optional.

The goal is for developers to adopt these new component properties functions. In so doing, common properties can be defined directly, and it will be possible to configure all exporters by means of the same set of properties. This means richer scripting and greater opportunities for custom user interfaces.

The Component Manager defines new optional standard selectors for component properties which can be implemented by all components, regardless of type, that are written to work with the component properties mechanism.Existing components can adopt these new functions on their own schedule. The Component Manager allows callers to discover safely whether components implement the new selectors and perform operations according to existing mechanisms if the new support is unavailable.

UI Examples with QuickTime Dialogs as Sheets

The following code shows how to use an effect and standard compression dialog as a sheet.

```
- (QTAtomContainer) getEffectSettingsFromSheetOnWindow:(NSWindow *)parentWindow
{
  QTAtomContainer                fxList = NULL;
  long                           minSrcs = 2;
  long                           maxSrcs = 2;
  QTEffectListOptions            lOpts = 0;
  QTEffectListOptions            dOpts = pdOptionsDisplayAsSheet;
  QTParameterDialog              paramDlg;
  QTEventLoopDescriptionRecord   eld;
  QTAtomContainer                fxDesc = nil;
  OSErr                          myErr;

  myErr = QTGetEffectsList ( &fxList, minSrcs, maxSrcs, lOpts );
  myErr = QTNewAtomContainer ( &fxDesc );
  myErr = QTCreateStandardParameterDialog ( fxList, fxDesc,
                                            dOpts, &paramDlg );

  eld.recordSize = sizeof(eld);
  eld.windowRefKind = kEffectParentWindowCarbon;
  eld.parentWindow = [parentWindow windowRef];
  eld.eventTarget = NULL;

  QTStandardParameterDialogDoAction(paramDlg, pdActionRunInEventLoop, &eld);

  QTDismissStandardParameterDialog(gParamDlg);
  return fxList;
}

- (NSData *) getCustomAudioCompressionSettingsFromSheetOnWindow:(NSWindow
*)parentWindow
{
    ComponentInstance   ci;
    Handle              settingsHndl = NULL;
    ComponentResult     result = noErr;
    SCWindowSettings    windowSettings;
    NSData              *settingsAsData = nil;

    ci = OpenDefaultComponent(StandardCompressionType,
                              StandardCompressionSubTypeSound);

    // Tell the standard compression component to show the settings panel
    // as a sheet
```

```
    windowSettings.size = sizeof(SCWindowSettings);
    windowSettings.windowRefKind = scWindowRefKindCarbon;
    windowSettings.parentWindow = [parentWindow windowRef];
    (void)SCSetInfo(ci, scWindowOptionsType, &windowSettings);

    // Get compression settings from the user.
    result = SCRequestImageSettings(ci);

    // Get the settings back from the standard compression component
    result = SCGetInfo(ci, scSettingsStateType, &settingsHndl);
    if (NULL != settingsHndl) {
        HLock(settingsHndl);
        settingsAsData = [NSData dataWithBytes:*settingsHndl
                                        length:GetHandleSize(settingsHndl)];
        DisposeHandle(settingsHndl);
    }

    CloseComponent(ci);
    return settingsAsData;
}
```

New Movie Property Functions

QuickTime 6.4 includes five movie property functions that are similar in purpose to the component functions described in "New Component Properties Functions" (page 21):

- `QTGetMoviePropertyInfo` (page 70) returns information about the properties of a movie.
- `QTGetMovieProperty` (page 69) returns the value of a specific movie property.
- `QTSetMovieProperty` (page 79) sets the value of a specific movie property.
- `QTAddMoviePropertyListener` (page 56) installs a callback to monitor a movie property.
- `QTRemoveMoviePropertyListener` (page 78) removes a movie property monitoring callback.

New IIDC Digitizer Functions

Many IEEE-1394-based digital cameras and webcams now support Instrumentation and Industrial Control (IIDC) features that can be accessed by software.

QuickTime 6.4 includes six new functions to communicate with video digitizers that have a subtype of `vdSubtypeIIDC ('iidc')`:

- `VDIIDCGetFeatures` (page 83) places atoms in a QuickTime atom container that specify the current capabilities of a camera and the state of its IIDC features.
- `VDIIDCGetFeaturesForSpecifier` (page 83) places atoms in a QuickTime atom container that specify the current state of a single camera IIDC feature or group of features.
- `VDIIDCSetFeatures` (page 86) changes the state of a camera's IIDC features.

- VDIIDCGetDefaultFeatures (page 82) places atoms in a QuickTime atom container that specify the default capabilities and default state of a camera's IIDC features.
- VDIIDCGetCSRData (page 81) reads a camera's CSR registers directly.
- VDIIDCSetCSRData (page 85) writes to a camera's CSR registers directly.

Several data structures that support these functions are new in QuickTime 6.4:

- VDIIDCFeatureAtomTypeAndID (page 91) provides content for the vdIIDCAtomTypeFeatureAtomTypeAndID atom type, discussed in "Type And ID Atoms" (page 25). It contains general information about an IIDC feature and specifies the atom that holds that feature's current settings.
- The VDIIDCFeatureSettings (page 93) structure provides content for the vdIIDCAtomTypeFeatureSettings atom type, discussed in "IIDC Settings Atoms" (page 25). It contains the VDIIDCFeatureCapabilities and VDIIDCFeatureState structures, which are used to describe various IIDC camera features.
- VDIIDCFeatureCapabilities (page 92) provides IIDC feature capabilities information for the VDIIDCFeatureSettings structure.
- VDIIDCFeatureState (page 94) provides IIDC feature state information for the VDIIDCFeatureSettings structure.
- VDIIDCTriggerSettings (page 98) provides content for the vdIIDCAtomTypeTriggerSettings atom type. It contains the VDIIDCTriggerCapabilities and VDIIDCTriggerState structures, which contain information about a camera's trigger capabilities and state.
- VDIIDCTriggerCapabilities (page 96) provides trigger capabilities information for the VDIIDCTriggerSettings structure.
- VDIIDCTriggerState (page 98) provides trigger state information for the VDIIDCTriggerSettings structure.
- VDIIDCFocusPointSettings (page 95) provides content for the vdIIDCAtomTypeFocusPointSettings atom type. It contains focus point data.
- VDIIDCLightingHintSettings (page 95) provides content for the vdIIDCAtomTypeLightingHintSettings atom type. It contains lighting hint data.

IIDC Atoms

The information about IIDC features that a camera might support is contained in a hierarchy of new big-endian QuickTime atom types:

- At the top level, the QuickTime atom container passed into VDIIDCSetFeatures or returned by VDIIDCGetFeatures, VDIIDCGetDefaultFeatures, or VDIIDCGetFeaturesForSpecifier contains a set of atoms of type vdIIDCAtomTypeFeature, one for each feature of the camera being interrogated.
- Each atom of type vdIIDCAtomTypeFeature contains one atom of type vdIIDCAtomTypeFeatureAtomTypeAndID. This atom contains a data structure of type VDIIDCFeatureAtomTypeAndID (page 91), which conveys general information about the feature and specifies the type and ID of the atom that conveys that feature's current settings.

- Each atom of type `vdIIDCAtomTypeFeature` also contains a Settings atom. This may be an atom of type `vdIIDCAtomTypeFeatureSettings`, `vdIIDCAtomTypeTriggerSettings`, `vdIIDCAtomTypeFocusPointSettings`, or `vdIIDCAtomTypeLightingHintSettings`, each of which stores a particular class of camera settings.

Details of the new atom types are given in the next sections. To determine the number of features in a particular atom container, you can use this code:

```
SInt16 featureCount = QTCountChildrenOfType(container,
    kParentAtomIsContainer, vdIIDCAtomTypeFeature);
```

IIDC Feature Atoms

The top-level atoms in the IIDC atom hierarchy are Feature atoms, of type `vdIIDCAtomTypeFeature`, the value of which is `'feat'`. Zero or more of these atoms are present in the atom container that is passed to the `VDIIDCSetFeatures` function and returned by the `VDIIDCGetFeatures`, `VDIIDCGetDefaultFeatures`, and `VDIIDCGetFeaturesForSpecifier` functions.

Each Feature atom is a container for atoms that convey information about one IIDC feature of the camera being interrogated or set.

Each Feature atom has two child atoms. The first is a Type and ID atom, of type `vdIIDCAtomTypeFeatureAtomTypeAndID`, described in the next section. The Type and ID atom identifies a second child atom of the `vdIIDCAtomTypeFeature` atom, which contains the feature's current or default setting information. These atoms are described in "IIDC Settings Atoms" (page 25)below.

Type And ID Atoms

Each feature atom contains one Type and ID atom, of type `vdIIDCAtomTypeFeatureAtomTypeAndID` (`'t&id'`). Its ID is 1. The contents of this atom is a `VDIIDCFeatureAtomTypeAndID` (page 91) data structure. This structure contains the type of the feature, a type that identifies the general category of the feature (image control, color control, mechanics, or triggering), the feature's human-readable name, and the type and ID of the Settings atom that contains the feature's settings.

IIDC Settings Atoms

Most IIDC features can be expressed by `VDIIDCFeatureSettings` (page 93) data structures. Three, however, require `VDIIDCTriggerSettings` (page 98), `VDIIDCFocusPointSettings` (page 95), or `VDIIDCLightingHintSettings` (page 95) data structures, as shown in the following listing:

```
vdIIDCFeatureHue                = 'hue ', // Uses VDIIDCFeatureSettings
vdIIDCFeatureSaturation         = 'satu', // Uses VDIIDCFeatureSettings
vdIIDCFeatureSharpness          = 'shrp', // Uses VDIIDCFeatureSettings
vdIIDCFeatureBrightness         = 'brit', // Uses VDIIDCFeatureSettings
vdIIDCFeatureGain               = 'gain', // Uses VDIIDCFeatureSettings
vdIIDCFeatureIris               = 'iris', // Uses VDIIDCFeatureSettings
vdIIDCFeatureShutter            = 'shtr', // Uses VDIIDCFeatureSettings
vdIIDCFeatureExposure           = 'xpsr', // Uses VDIIDCFeatureSettings
vdIIDCFeatureWhiteBalanceU      = 'whbu', // Uses VDIIDCFeatureSettings
vdIIDCFeatureWhiteBalanceV      = 'whbv', // Uses VDIIDCFeatureSettings
vdIIDCFeatureGamma              = 'gmma', // Uses VDIIDCFeatureSettings
vdIIDCFeatureTemperature        = 'temp', // Uses VDIIDCFeatureSettings
vdIIDCFeatureZoom               = 'zoom', // Uses VDIIDCFeatureSettings
```

```
vdIIDCFeatureFocus              = 'fcus', // Uses VDIIDCFeatureSettings
vdIIDCFeaturePan                = 'pan ', // Uses VDIIDCFeatureSettings
vdIIDCFeatureTilt               = 'tilt', // Uses VDIIDCFeatureSettings
vdIIDCFeatureOpticalFilter      = 'opft', // Uses VDIIDCFeatureSettings
vdIIDCFeatureTrigger            = 'trgr', // Uses VDIIDCTriggerSettings
vdIIDCFeatureCaptureSize        = 'cpsz', // Undefined settings
vdIIDCFeatureCaptureQuality     = 'cpql', // Undefined settings
vdIIDCFeatureFocusPoint         = 'fpnt', // Uses VDIIDCFocusPointSettings
vdIIDCFeatureEdgeEnhancement    = 'eden'  // Uses VDIIDCFeatureSettings
vdIIDCFeatureLightingHint       = 'lhnt'  // Uses VDIIDCLightingHintSetting
```

The types and IDs of the Settings atoms that contain these data structures are the following:

```
// Atom type and ID that contains the VDIIDCFeatureSettings struct
vdIIDCAtomTypeFeatureSettings       = 'fstg',
vdIIDCAtomIDFeatureSettings         = 1

// Atom type and ID that contains the VDIIDCTriggerSettings struct
vdIIDCAtomTypeTriggerSettings       = 'tstg',
vdIIDCAtomIDTriggerSettings         = 1

// Atom type and ID that contains the VDIIDCFocusPointSettings struct
vdIIDCAtomTypeFocusPointSettings     = 'fpst',
vdIIDCAtomIDFocusPointSettings       = 1

// Atom type and ID that contains the VDIIDCLightingHintSetting struct
vdIIDCAtomTypeLightingHintSettings   = 'lhst',
vdIIDCAtomIDLightingHintSettings     = 1
```

A Typical IIDC Atom Hierarchy

To take a specific example, suppose you called `VDIIDCGetFeatures`, passing an instance of a video digitizer of subtype `vdSubTypeIIDC`. You might receive back a QuickTime atom container in which you find the following atoms:

```
vdIIDCAtomTypeFeature    = 'feat'

    vdIIDCAtomTypeFeatureAtomTypeAndID    = 't&id'
    vdIIDCAtomIDFeatureAtomTypeAndID      = 1
        feature     = vdIIDCFeatureShutter            // shutter feature
        atomType    = vdIIDCAtomTypeFeatureSettings
        atomID      = 1

    vdIIDCAtomTypeFeatureSettings    = 'fstg'
    vdIIDCAtomIDFeatureSettings      = 1


vdIIDCAtomTypeFeature    = 'feat'

    vdIIDCAtomTypeFeatureAtomTypeAndID    = 't&id'
    vdIIDCAtomIDFeatureAtomTypeAndID      = 1
        feature     = vdIIDCFeatureFocus              // focus feature
        atomType    = vdIIDCAtomTypeFeatureSettings
        atomID      = 1

    vdIIDCAtomTypeFeatureSettings    = 'fstg'
    vdIIDCAtomIDFeatureSettings      = 1
```

```
vdIIDCAtomTypeFeature    = 'feat'

    vdIIDCAtomTypeFeatureAtomTypeAndID    = 't&id'
    vdIIDCAtomIDFeatureAtomTypeAndID      = 1
        feature     = vdIIDCFeatureTrigger            // trigger feature
        atomType    = vdIIDCAtomTypeTriggerSettings
        atomID      = 1

    vdIIDCAtomTypeTriggerSettings    = 'tstg'
    vdIIDCAtomIDFTriggerSettings     = 1
```

In this example, the `VDIIDCGetFeatures` function tells you that the camera served by the video digitizer component passed to it reports on three IIDC functions: shutter, focus, and trigger. Data on the shutter and focus features can be retrieved from `VDIIDCFeatureSettings` structures in `vdIIDCAtomTypeFeatureSettings` atoms. Data on the trigger feature can be retrieved from a `VDIIDCTriggerSettings` structure in a `vdIIDCAtomTypeTriggerSettings` atom.

New Sound Function

QuickTime 6.4 includes a new sound function to identify the audio device used by a video output component. It is `QTVideoOutputCopyIndAudioOutputDeviceUID` (page 80).

New Offset TimeBase Functions

QuickTime 6.4 includes two offset timebase functions, so custom media handlers can implement media latency.

The new functions are `SetTimeBaseOffsetTimeBase` (page 81) and `GetTimeBaseMasterOffsetTimeBase` (page 42).

Changes to Text Drawing

In QuickTime 6.4, text drawing now uses the ATSUI text drawing engine instead of TextEdit. This facilitates the vertical drawing of text––for example, tx3g for Japanese rendering of vertical text. Internally, this means that content authors should be aware of the subtle differences in text rendering, in that all text is now converted to Unicode to be drawn by ATSUI. ATSUI may position characters slightly differently than TextEdit because QuickDraw is no longer used. The outline and shadow styles are no longer supported.

ATSUI is used only on the Macintosh platform. The corresponding drawing engine for Windows is Uniscribe.

Encoding Text Changes

With QuickTime 6.4, the internal behavior for encoding text has changed. All text is now converted to Unicode (if it is not already in Unicode) instead of 8-bit ASCII. This ensures better rendering of Unicode characters.

New Release of QuickTime for Java

The release of QuickTime 6.4 includes QuickTime for Java 1.4.1, discussed in this section. The Software Development Kit (SDK) is included in Mac OS X version 10.3. You can download it from `http://developer.apple.com/sdk/index.html`.

Goals

The goal of this release is to provide QuickTime for Java developers with a new version of QTJava that works with both JDK 1.3 and JDK 1.4.1 on Mac OS X. The current version of QTJava supports JDK 1.4.1, but only on Windows. The intent of this release is to provide a minimal set of functionality across all platforms.

QuickTime for Java 1.4.1 supports this goal by introducing a new `quicktime.app.view` package, adding other packages, and deprecating some existing packages. But note that all classes that are directly bound to the underlying QuickTime API, including all `quicktime.std` packages, remain the same and are supported. Developers will still have access to all the QuickTime features through Java using these bindings.

Because many features of QTJava are already available in standard Java, this refinement and simplification of the QTJava architecture has become necessary.

Hierarchy For All New Packages

This section describes the new packages and interfaces in QuickTime for Java 1.4.1, including the `quicktime.app.view` package, its class hierarchy and its interface hierarchy.

Package Hierarchies

```
quicktime.app.time
quicktime.app.view
```

Class Hierarchy

```
class java.lang.Object
    class quicktime.app.view. GraphicsImporterDrawer (implements
                                  quicktime.app.view.Presentable)
    class quicktime.app.view. MoviePlayer (implements
                                  quicktime.app.view.DrawingNotifier,
                                  quicktime.app.view.Presentable,
                                  quicktime.app.time. Timeable
    class quicktime.app.view. QTFactory
    class quicktime.app.view. QTImageProducer (implements
```

```
                                java.awt.image.ImageProducer)
    class quicktime.app.time. Tasking (implements
                                quicktime.app.time.Taskable)
    class quicktime.app.time. TaskAllMovies
    class quicktime.app.time. TaskThread (implements
                                java.lang.Runnable)
```

Interface Hierarchy

```
interface quicktime.app.view. QTComponent
interface quicktime.app.view. QTJComponent
interface quicktime.app.time. Timeable
```

Migrating Existing Code to the New Classes

To take advantage of the new functionality in QuickTime for Java 1.4.1, developers will need to migrate their code to use the new classes and interfaces. This means rewriting existing application code.

New sample code, however, is listed below. It illustrates the usage of the new QTJava classes and methods. These new classes are backward compatible, so that you can use the same new classes on the Java 1.3 and 1.4.1 virtual machines.

SDK Examples That Work with JDK 1.4.1

The following SDK examples, accompanying this release, will work with JDK 1.4.1 and QuickTime for Java 1.4.1. You can build and compile these examples in Xcode on Mac OS X version 10.3:

```
AddTextMovie
PlaySound
TimeCallbackDemo
Applets
CustomMedia
KeyboardController
PlayTune
TimeCode
AudioBroadcaster
DetachedController
ImageFile
MovieCallbacks
SoundMemRecord
ImageProducing
MovieTextFinder
QTVector
SoundMeter
VRInteraction
ImportExport
Music
QTtoJavaImage
SoundRecord
CreatePictFile
DukeMovie
JavaDrawing
PlayMovie
```

The following SDK examples will not work with JDK 1.4.1 and QuickTime for Java 1.4.1. This is because these examples use presenters, compositing, or sprites, which are not supported.

```
JavaSprites
GroupDrawing
BouncingSprites
DraggingSprites
CurvesDemo
GraphicsExport
MediaPresenter
SlideShow
DraggingSpritesApplet
DrawableBroadcaster
WiredSprites
ImageCompositing
QTButtonDemo
QTEffects
Transitions
TextDemo
```

The following SDK examples are not yet implemented in QuickTime for Java 1.4.1:

```
SGCapture
TimeSlaving
SGCapture2Disk
```

Packages Not Supported in this Release

The following packages are not supported in QuickTime for Java 1.4.1:

```
quicktime.app.actions
quicktime.app.anim
quicktime.app.audio
quicktime.app.display
quicktime.app.event
quicktime.app.image
quicktime.app.players
quicktime.app.sg
quicktime.app.spaces
quicktime.app.ui
```

This means that such capabilities as compositing, animation with sprites and a sprite world, and the usage of spaces and controllers are not supported using the new classes in QuickTime for Java 1.4.1.

Although sequence grabbing is currently not supported in QuickTime for Java 1.4.1, it may be provided in future releases.

For developers, note that the `QTDrawable` interfaces, as well as `QTCanvas` and `JQTCanvas`, are not supported in QuickTime for Java 1.4.1. These interfaces and classes were part of the `quicktime.app.display` package and dealt with displaying QuickTime content in a Java AWT or Swing frame.

Note that `quicktime.app.time` is still supported, including classes such as `TaskThread`, `Tasking`, `TaskAllMovies`, and the `Timeable` interface.

The quicktime.app.view Package

The new `quicktime.app.view` package now deals with issues of display and the content that QuickTime can draw into Java. The underlying implementation is hidden: QTJava now provides a Java AWT or Java Swing component, based on what you want to use it for. This means that you won't need to access `QTCanvas` or subclass the component and add your own functionality as you want and draw into it. The component will already be constructed for you when you want to display a movie, an image, or a movie with a controller.

This new package includes the following classes:

- `QTFactory` (which has been moved from the `quicktime.app` package)
- `QTImageProducer` (moved from the `quicktime.app.image` package)
- `MoviePlayer` (moved from the `quicktime.app.players` package)
- `GraphicsImporterDrawer` (moved from the `quicktime.app.image` package)

The package also includes two new interfaces:

- `QTComponent`
- `QTJComponent`

The methods for these new interfaces and classes in the `quicktime.app.view` package are described in the Javadoc documentation accompanying QuickTime for Java 1.4.1.

The `QTFactory` class includes two new factory methods in addition to those already supported. These methods are

- `makeQTComponent`
- `makeQTJComponent`

Developers can use these new methods to create either `QTComponent` or `QTJComponent` objects. For example, you can use the `makeQTComponent` factory method, passing it a movie object, and it will return an instance of a `QTComponent` interface. Then, if you want, you can directly use this component, which is a regular AWT component, and add it to a Frame and display it.

`QTComponent` and `QTJComponent` provide you with interfaces to set and get the existing movie or image, whichever is displayed in the component. You pass in `null` as a parameter in order to remove an existing image or movie controller.

`QTComponent` has three set and get methods:

- `setMovie` and getMovie
- `setMovieController` and getMovieController
- `setImage` and getImage

The `QTComponent` and `QTJComponent` interfaces are implemented by the underlying component classes.

You can create a `QTComponent` object with an existing movie or image in it, and if you want to change the movie or image displayed, you can use the set methods to change the movie or image displayed in the component. If you just want to remove it, while cleaning up, you can pass in a `null` parameter.

To create a movie controller, you need to pass in a movie first. You create a movie object, pass it in to a movie controller, and create a movie controller object. Then you take this movie controller object and pass it to the `makeQTComponent` QuickTime factory method. This will return a Java AWT component with a movie and a controller attached and visible. This will be a regular QuickTime movie with a controller attached to it.

You use the `makeQTJComponent` method to create a `QTJComponent`, which is a Swing component. To display a movie inside a Swing component, you use the `QTFactory.makeQTJComponent` method.

Migrating Old QTJava Code to New QTJava Code

The following example illustrate some ways that you can migrate existing QuickTime for Java code—for example, code that uses `QTCanvas`—to the new classes and methods supported in release 1.4.1.

```
// Old Code :

// Movie Example Using QTCanvas :
// create movie from QTFile

QTFile qtfile = new QTFile (moviePath);
Movie mov = Movie.fromFile(qtfile);
MovieController mc = new MovieController(mov);
QTPlayer qtp = new QTPlayer(mc);

// create canvas and set the client
QTCanvas qtc = new QTCanvas ();
frame.add(qtc);
qtc.setClient(qtp, true);

frame.show();

// New Code :

// Movie Example Using QTFactory methods :
// create movie from QTFile
QTFile qtfile = new QTFile (moviePath);
Movie mov = Movie.fromFile(qtfile);
MovieController mc = new MovieController(mov);

// create component using factory methods
QTComponent movComp = QTFactory.makeQTComponent(mc);
frame.add((Component)movComp);
frame.show();


// Swing Example :

// Old Code :
// create movie from QTFile
QTFile qtfile = new QTFile (moviePath);
Movie mov = Movie.fromFile(qtfile);
MoviePlayer movPlayer = new MoviePlayer(mov);

// create JQTCanvas and set client
JQTCanvas jcanvas = new JQTCanvas();
jframe.getConentPane().add(jcanvas);
jcanvas.setClient(movPlayer, true);
```

```
jframe.show();


// New Code :

// create movie from QTFile
QTFile qtfile = new QTFile (moviePath);
Movie mov = Movie.fromFile(qtfile);
MoviePlayer movPlayer = new MoviePlayer(mov);

// create component using factory methods
QTJComponent  movComponent = QTFactory.makeQTJComponent(movPlayer);
jframe.getConentPane().add((JComponent)movComponent);

jframe.show();
```

QuickTime 6.4 API Reference

Functions

The functions new to the QuickTime 6.4 API are documented alphabetically in this section.

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

Declared In
`Movies.h`

AttachTimeBaseToCurrentThread

Attaches a time base to the current thread.

```
OSErr AttachTimeBaseToCurrentThread (TimeBase tb);
```

Parameters

tb

A time base.

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

Version Notes
Introduced in QuickTime 6.4.

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

Declared In
`Movies.h`

ClockGetRateChangeConstraints

Obtains minimum and maximum delays that a clock could introduce during a rate change.

```
ComponentResult ClockGetRateChangeConstraints (
    ComponentInstance    aClock,
    TimeRecord           *minimum,
    TimeRecord           *maximum );
```

Parameters

aClock

Specifies the clock for the operation. Applications obtain this identifier from `OpenComponent`.

minimum

A pointer to a `TimeRecord` structure that the clock will update with the minimum delay introduced during a rate change. You can pass `NIL` if you do not want to receive this information.

maximum

A pointer to a `TimeRecord` structure that the clock will update with the maximum delay introduced during a rate change. You can pass `NIL` if you do not want to receive this information.

Return Value
See "Error Codes" in the QuickTime API Reference. Returns `noErr` if there is no error. Returns `badComponentSelector` if the component does not support the call.

Version Notes
Introduced in QuickTime 6.4.

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

Declared In
`QuickTimeComponents.h`

ClockGetTimeForRateChange

Obtains the current rate change time according to a specific clock, a preferred time, and a safe increment duration.

```
ComponentResult ClockGetTimesForRateChange (
    ComponentInstance    aClock,
    Fixed                fromRate,
    Fixed                toRate,
    TimeRecord           *currentTime,
    TimeRecord           *preferredTime,
    TimeRecord           *safeIncrementForPreferredTime );
```

Parameters

aClock

Specifies the clock for the operation. Applications obtain this identifier from `OpenComponent`.

fromRate

The clock rate you are starting from.

toRate

The clock rate you are going to.

currentTime

A pointer to a `TimeRecord` structure that the clock will update with the current rate change time. You can pass `NIL` if you do not want to receive this information.

preferredTime

A pointer to a `TimeRecord` structure that the clock will update with the next clock time at which the clock would prefer dependent time bases to perform the rate change. You can pass `NIL` if you do not want to receive this information.

safeIncrementForPreferredTime

A pointer to a `TimeRecord` structure that the clock will update with the increment between preferred times for the rate change. If this increment is nonzero, multiples of it may be added to the `preferredTime` value to calculate future preferred times. You can pass `NIL` if you do not want to receive this information.

Return Value

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

Discussion

The `preferredTime` parameter indicates a better choice than the current time to have a rate transition `fromRate` to `toRate`. But the clock cannot expect the interesting time to be the one used when the rate changes. The `safeIncrementForPreferredTime` indicates that the preferred time can be incremented by this value (and any multiple of it) and still be safe to use.

Version Notes

Introduced in QuickTime 6.4.

Availability

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

ConvertDataRefToMovieDataRef

Converts a piece of data in a storage location to a movie file format and stores it in another storage location, supporting a user settings dialog box for import operations.

```
OSErr ConvertDataRefToMovieDataRef (
    Handle               inputDataRef,
    OSType               inputDataRefType,
    Handle               outputDataRef,
```

```
    OSType              outputDataRefType,
    OSType              creator,
    long                flags,
    ComponentInstance   userComp,
    MovieProgressUPP    proc,
    long                refCon );
```

Parameters

inputDataRef

A data reference that specifies the storage location of the source data.

inputDataRefType

The type of the input data reference.

outputDataRef

A data reference that specified the storage location to receive the converted data.

outputDataRefType

The type of the output data reference.

creator

The creator type of the output storage location.

flags

Flags that control the operation of the dialog box.

`createMovieFileDeleteCurFile`

Indicates whether to delete an existing file. If you set this flag to 1, the Movie Toolbox deletes the file (if it exists) before converting the new movie file. If you set this flag to 0 and the file specified by the `inputDataRef` and `outputDataRef` parameters already exists, the Movie Toolbox uses the existing file. In this case, the toolbox ensures that the file has both data.

`movieToFileOnlyExport`

If this bit is set and the `showUserSettingsDialog` bit is set, the Save As dialog box restricts the user to those file formats that are supported by movie data export components.

`movieFileSpecValid`

If this bit is set and the `showUserSettingsDialog` bit is set, the name of the `outputDataRef` parameter is used as the default name of the exported file in the Save As dialog box.

`showUserSettingsDialog`

Controls whether the user settings dialog box for the specified import operation can appear. Set this flag to 1 to display the user settings dialog box.

userComp

An instance of a component to be used for converting the movie data.

proc

A progress callback function; see `MovieProgressProc` in the QuickTime API Reference.

refCon

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

Return Value

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

Discussion
This function converts a piece of data in a storage location into a movie and stores into another storage location. Both the input and the output storage locations are specified through data references. If the storage location is on a local file system, the file will have the specified creator. If specified as such in the flags, the function displays a dialog box that lets the user to choose the output file and the export type. If an export component (or its instance) is specified in `userComp`, it will be used for the conversion operation.

Version Notes
Introduced in QuickTime 6.4.

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

Declared In
`Movies.h`

ConvertMovieToDataRef

Converts a specified movie (or a single track within a movie) into a specified file format and stores it in a specified storage location.

```
OSErr ConvertMovieToDataRef (
    Movie               theMovie,
    Track               onlyTrack,
    Handle              dataRef,
    OSType              dataRefType,
    OSType              fileType,
    OSType              creator,
    long                flags,
    ComponentInstance   userComp );
```

Parameters

theMovie

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

onlyTrack

The track in the source movie, if you want to convert only a single track.

dataRef

A data reference that specifies the storage location to receive the converted movie data.

dataRefType

The type of data reference. This function currently supports only alias data references.

fileType

The Mac OS file type of the storage location, which determines the export format.

creator

The creator type of the storage location.

flags

Flags (see below)that control the operation of the dialog box.

`showUserSettingsDialog`

If this bit is set, the Save As dialog box will be displayed to allow the user to choose the type of file to export to, as well as the file name to export to.

`movieToFileOnlyExport`

If this bit is set and the `showUserSettingsDialog` bit is set, the Save As dialog box restricts the user to those file formats that are supported by movie data export components.

`movieFileSpecValid`

If this bit is set and the `showUserSettingsDialog` bit is set, the name field of the `dataRef` parameter is used as the default name of the exported file in the Save As dialog box.

userComp

An instance of the component to be used for converting the movie data.

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

Discussion
If the storage location is on a local file system, the file will have the specified file type and the creator. If specified as such in the flags, the function displays a dialog box that lets the user choose the output file and the export type. If an export component (or its instance) is specified in the `userComp` parameter, it will be used to perform the conversion operation.

Version Notes
Introduced in QuickTime 6.4.

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

Declared In
`Movies.h`

ConvertTimeToClockTime

Converts a time record in a time base to clock time.

```
void ConvertTimeToClockTime ( TimeRecord *time );
```

Parameters

time

The `TimeRecord` structure to be converted. It must contain a valid time base; otherwise it remains untouched.

Discussion
The result of this call has no meaning it the time base rate is 0.

Version Notes
Introduced in QuickTime 6.4.

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

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

Declared In
`Movies.h`

DetachTimeBaseFromCurrentThread

Detaches a time base from the current thread.

```
OSErr DetachTimeBaseFromCurrentThread (TimeBase tb);
```

Parameters

tb

A time base.

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

Version Notes
Introduced in QuickTime 6.4.

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

Declared In
`Movies.h`

EnterMoviesOnThread

Indicates that the client will be using QuickTime on the current thread.

```
OSErr EnterMoviesOnThread (UInt32 inFlags);
```

Parameters

inFlags

Flags indicating how the executing thread will be using QuickTime. Setting the thread mode is a convenience provided by this function. Pass 0 for the default options.

`kQTEnterMoviesFlagDontSetComponentsThreadMode`

Public state. Value is `1L << 0`. By default, this function sets the current Component Manager thread mode according to `kCSAcceptThreadSafeComponentsOnlyMode`. By setting the `kQTEnterMoviesFlagDontSetComponentThreadMode` flag, no change to the thread mode will be made, leaving it as it was before the call to this function.

Return Value

See "Error Codes" in the QuickTime API Reference. Returns `noErr` if there is no error. This function returns an appropriate operating system or QuickTime error if the operation couldn't be completed. This might occur because a second call on the thread was made that used incompatible flags (for example, the first call required a shared state but a subsequent call required a private state).

Discussion

This function is analogous to `EnterMovies`. It initializes QuickTime and prepares QuickTime for calls from its thread. Unlike `EnterMovies`, this function allows the client to indicate if its access to QuickTime requires sharing of QuickTime state with the main thread. The default is to maintain a private state.

Your application should call this function on threads you create. Calling it on the main thread should be done if QuickTime will be used from the main thread.

If a client doesn't call this function on a spawned preemptive thread and then makes a QuickTime call (including `EnterMovies`), the global QuickTime state will be shared with the main (or application) thread. This behavior ensures compatibility with current applications accessing QuickTime from multiple threads.

The first call to this function will change the components thread mode unless the `kQTEnterMoviesFlagDontSetComponentsThreadMode` flag is passed. All subsequent calls will leave the components thread mode unaffected.

Version Notes

Introduced in QuickTime 6.4.

Availability

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

Declared In

`Movies.h`

ExitMoviesOnThread

Indicates to QuickTime that the client will no longer be using QuickTime on the current thread.

```
OSErr ExitMoviesOnThread (void);
```

Return Value

See "Error Codes" in the QuickTime API Reference. Returns `noErr` if there is no error. Returns an appropriate operating system or QuickTime error if the operation couldn't be completed. This might occur because a previous call to `EnterMoviesOnThread` was not made.

Discussion
This function should be called before exiting from a spawned thread that uses QuickTime. It undoes the setup performed by `EnterMoviesOnThread`. Each call to `EnterMoviesOnThread` should be matched with a call to this function. This function should not be called on a thread without a previous call to `EnterMoviesOnThread`.

After the last call to this function is made on a thread, subsequent calls to QuickTime functions without calling `EnterMoviesOnThread` first will result in threads sharing the main thread's state just as though the application didn't use `EnterMoviesOnThread` or this function. This behavior ensures compatibility. Thus callers should bracket all QuickTime calls on secondary threads between an initial call to `EnterMoviesOnThread` and final call to this function.

If you do not call this function, certain cleanup may not occur, potentially causing resource leaks.

Version Notes
Introduced in QuickTime 6.4.

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

Declared In
`Movies.h`

GetMovieRateChangeConstraints

Returns the minimum and maximum delay you can get when a movie's rate is changed.

```
void GetMovieRateChangeConstraints (
    Movie         theMovie,
    TimeRecord    *minimumDelay,
    TimeRecord    *maximumDelay );
```

Parameters

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

`minimumDelay`
A pointer to a `TimeRecord` structure. The function updates this structure to contain the minimum delay when a rate change happens.

`maximumDelay`
A pointer to a `TimeRecord` structure. The function updates this structure to contain the maximum delay when a rate change happens.

Discussion
If the time base master clock of the movie is changed, this function must be called again to reflect the current constraints.

Version Notes
Introduced in QuickTime 6.4.

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

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

Declared In
Movies.h

GetTimeBaseMasterOffsetTimeBase

Allows an offset time base to retrieve the master time base it is attached to.

```
TimeBase GetTimeBaseMasterOffsetTimeBase ( TimeBase
tb );
```

Parameters

tb

An offset time base.

Return Value
The master time base for the offset time base passed in *tb*. Returns NIL if tb does not contain an offset time base.

Version Notes
Introduced in QuickTime 6.4.

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

Declared In
`Movies.h`

GetTimeBaseRateChangeStatus

Lets a time base client determine the time base's last rate change status.

```
void GetTimeBaseRateChangeStatus (
    TimeBase          tb,
    TimeScale         scale,
    Fixed             *rateChangedTo,
    TimeBaseStatus    *flags,
    TimeRecord        *rateChangeTimeBaseTime,
    TimeRecord        *rateChangeClockTime,
    TimeRecord        *currentClockTime );
```

Parameters

tb

A pointer to a `TimeBaseRecord` structure.

scale

The scale to use for the returned time values. Pass 0 to retrieve the time in the preferred time scale of the time base.

rateChangedTo

The rate value changed to. Clients may pass `NIL` if they do not want to receive this information.

flags

A pointer to a flag that will be returned when the clock is waiting for a future time to start moving while its rate is nonzero. When set, the unpinned time will return a negative value telling how far you are from the real start time. Clients may pass `NIL` if they do not want to receive this information.

`rateChangeTimeBaseTime`

The time base time when the rate changed. Clients may pass `NIL` if they do not want to receive this information.

`rateChangeClockTime`

The clock time when the rate changed. Clients may pass `NIL` if they do not want to receive this information.

`currentClockTime`

The current clock time value. Clients may pass `NIL` if they do not want to receive this information.

`timeBaseRateChanging`

The clock is waiting for a future time to start moving while its rate is nonzero. When set, the unpinned time will return a negative value telling how far you are from the real start time.

rateChangeTimeBaseTime

The time base time when the rate changed. Clients may pass `NIL` if they do not want to receive this information.

rateChangeClockTime

The clock time when the rate changed. Clients may pass `NIL` if they do not want to receive this information.

currentClockTime
The current clock time value. Clients may pass `NIL` if they do not want to receive this information.

Discussion
When the flag `timeBaseRateChanging` is returned, the amount of time left before the time base ticks is equal to `(rateChangeClockTime - currentClockTime)`.

Version Notes
Introduced in QuickTime 6.4.

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

Declared In
`Movies.h`

GetTimeBaseThreadAttachState

Determines whether a given time base is attached to a thread.

```
OSErr GetTimeBaseThreadAttachState (
    TimeBase    inTimeBase,
    Boolean     *outAttachedToCurrentThread,
    Boolean     *outAttachedToAnyThread );
```

Parameters
inTimeBase
A time base.

outAttachedToCurrentThread
A pointer to a `Boolean` that on exit is `TRUE` if the time base is attached to the current thread, `FALSE` otherwise.

outAttachedToAnyThread
A pointer to a `Boolean` that on exit is `TRUE` if the time base is attached to any thread, `FALSE` otherwise.

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

Version Notes
Introduced in QuickTime 6.4.

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

Declared In
`Movies.h`

GraphicsExportGetInputCGBitmapContext

Retrieves the `CGBitmapContext` that the graphics exporter is using as its input image.

```
ComponentResult GraphicsExportGetInputCGBitmapContext
(
    GraphicsExportComponent    ci,
    CGContextRef               *bitmapContextRef );
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

bitmapContextRef

A reference to the Core Graphics context.

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

Version Notes
Introduced in QuickTime 6.4.

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

Declared In
`ImageCompression.h`

GraphicsExportGetInputCGImage

Determines which Core Graphics `CGImage` is the source for a graphics export operation.

```
ComponentResult GraphicsExportGetInputCGImage (
    GraphicsExportComponent    ci,
    CGImageRef                 *imageRef );
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

imageRef

A reference to a Core Graphics image.

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

Version Notes
Introduced in QuickTime 6.4.

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

Declared In
`ImageCompression.h`

GraphicsExportSetInputCGBitmapContext

Sets the `CGBitmapContext` that the graphics exporter will use as its input image.

```
ComponentResult GraphicsExportSetInputCGBitmapContext
(
    GraphicsExportComponent    ci,
    CGContextRef               bitmapContextRef );
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

bitmapContextRef

A reference to the Core Graphics context.

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

Version Notes
Introduced in QuickTime 6.4.

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

Declared In
`ImageCompression.h`

GraphicsExportSetInputCGImage

Specifies a Core Graphics `CGImage` as the source for a graphics export operation.

```
ComponentResult GraphicsExportSetInputCGImage(
    GraphicsExportComponent    ci,
    CGImageRef                 imageRef );
```

Parameters

ci

The component instance that identifies your connection to the graphics exporter component.

imageRef

A reference to a CG image.

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

Version Notes
Introduced in QuickTime 6.4.

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

Declared In
`ImageCompression.h`

GraphicsImportCreateCGImage

Imports an image as a Core Graphics `CGImage`.

```
ComponentResult GraphicsImportCreateCGImage (
    GraphicsImportComponent    ci,
    CGImageRef                 *imageRefOut,
    UInt32                     flags );
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

imageRefOut

A reference to the CG image to be created.

flags

A flag that determines the settings to use.

`kGraphicsImportCreateCGImageUsingCurrentSettings`

Use the current settings.

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

Declared In
`ImageCompression.h`

GraphicsImportDoExportImageFileToDataRefDialog

Presents a dialog box that lets the user save an imported image in a foreign file format.

```
ComponentResult GraphicsImportDoExportImageFileToDataRefDialog
(
    GraphicsImportComponent    ci,
    Handle                     inDefaultDataRef,
    OSType                     inDefaultDataRefType,
    CFStringRef                prompt,
    ModalFilterYDUPP           filterProc,
    OSType                     *outExportedType,
    Handle                     *outExportedDataRef,
    OSType                     *outExportedDataRefType );
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

inDefaultDataRef

A data reference that specifies the default export location.

inDefaultDataRefType

The type of the data reference that specifies the default export location.

prompt

A reference to a `CFString` that contains the prompt text string for the dialog.

filterProc

A modal filter function; see `ModalFilterYDProc` in the QuickTime API Reference.

outExportedType

A pointer to an `OSType` entity where the type of the exported file will be returned.

outExportedDataRef
A pointer to an handle where the data reference to the exported file will be returned.

outExportedDataRefType
A pointer to an `OSType` entity where the type of the data reference that points to the exported file will be returned.

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

Discussion
This function presents a file dialog that lets the user to specify a file to which the exported data goes and a format into which image data is exported. By using data references, a long file name or Unicode file name can be used as a default file name as well as the name of the file into which the export data goes. This function is equivalent to `GraphicsImportDoExportImageFileDialog`.

Version Notes
Introduced in QuickTime 6.4.

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

Declared In
`ImageCompression.h`

GraphicsImportExportImageFileToDataRef

Saves an imported image in a foreign file format.

```
ComponentResult GraphicsImportExportImageFileToDataRef
(
    GraphicsImportComponent    ci,
    OSType                     fileType,
    OSType                     fileCreator,
    Handle                     dataRef,
    OSType                     dataRefType );
```

Parameters

ci
The component instance that identifies your connection to the graphics importer component.

fileType
The Mac OS file type for the new file, which determines the file format.

fileCreator
The creator type of the new file.

dataRef
A data reference that specifies a storage location to which the image is to be exported.

dataRefType
The type of the data reference.

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

Discussion
This function exports the imported image as a foreign file format specified by `fileType`. The exported data will be saved into a storage location specified by a data reference. You can use data reference functions to create a data reference for a file that has long or Unicode file name. This function is equivalent to `GraphicsImportExportImageFile`.

Version Notes
Introduced in QuickTime 6.4.

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

Declared In
`ImageCompression.h`

GraphicsImportGetDestinationColorSyncProfileRef

Retrieves a ColorSync profile from a graphics importer component.

```
GraphicsImportGetDestinationColorSyncProfileRef (
    GraphicsImportComponent   ci,
    CMProfileRef              *destinationProfileRef);
```

Parameters

ci
The component instance that identifies your connection to the graphics importer component.

destinationProfileRef
On return, a pointer to an opaque struct containing a ColorSync profile.

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

Version Notes
Introduced in QuickTime 6.4.

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

Declared In
`ImageCompression.h`

GraphicsImportGetGenericColorSyncProfile

Retrieves the generic colorsync profile for a graphics importer component.

```
GraphicsImportGetGenericColorSyncProfile (
    GraphicsImportComponent    ci,
    OSType                     pixelFormat,
    void                       *reservedSetToNULL,
    UInt32                     flags,
    Handle                     *genericColorSyncProfileOut );
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

pixelFormat

See "Pixel Formats" in the QuickTime API Reference.

reservedSetToNULL

Pass `NIL`.

flags

Currently not used.

genericColorSyncProfileOut

A handle to the the generic colorsync profile for the graphics importer.

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

Version Notes
Introduced in QuickTime 6.4.

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

Declared In
`ImageCompression.h`

GraphicsImportGetOverrideSourceColorSyncProfileRef

Retrieves the override ColorSync profile for a graphics importer component.

```
ComponentResult GraphicsImportGetOverrideSourceColorSyncProfileRef
(
    GraphicsImportComponent   ci,
    CMProfileRef              *outOverrideSourceProfileRef );
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

outOverrideSourceProfileRef

A pointer to an opaque struct containing a ColorSync profile.

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

Version Notes
Introduced in QuickTime 6.4.

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

Declared In
`ImageCompression.h`

GraphicsImportSaveAsPictureToDataRef

Creates a storage location that contains a QuickDraw picture for an imported image.

```
ComponentResult GraphicsImportSaveAsPictureToDataRef
(
    GraphicsImportComponent    ci,
    Handle                     dataRef,
    OSType                     dataRefType );
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

dataRef

A data reference that specifies a storage location to which the picture is to be saved.

dataRefType

The type of the data reference.

Return Value

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

Discussion

This function saves the imported image as a QuickDraw picture into a storage location specified through a data reference. You can use Data Reference Utilities to create a data reference for a file that has long or Unicode file name. This function is equivalent to `GraphicsImporterSaveAsPictureFile`.

Version Notes

Introduced in QuickTime 6.4.

Availability

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

Declared In

`ImageCompression.h`

GraphicsImportSaveAsQuickTimeImageFileToDataRef

Creates a storage location that contains a QuickTime image of an imported image.

```
ComponentResult GraphicsImportSaveAsQuickTimeImageFileToDataRef
(
    GraphicsImportComponent    ci,
    Handle                     dataRef,
    OSType                     dataRefType );
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

dataRef

A data reference that specifies a storage location to which the picture is to be saved.

dataRefType

The type of the data reference.

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

Discussion
This function saves the imported image as a QuickTime image into a storage location specified through a data reference. You can use data reference functions to create a data reference for a file that has long or Unicode file name. This function is equivalent to `GraphicsImportSaveAsQuickTimeImageFile`.

Version Notes
Introduced in QuickTime 6.4.

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

Declared In
`ImageCompression.h`

GraphicsImportSetDestinationColorSyncProfileRef

Sets the ColorSync profile for a graphics importer component.

```
ComponentResult GraphicsImportSetDestinationColorSyncProfileRef
(
    GraphicsImportComponent   ci,
    CMProfileRef              newDestinationProfileRef);
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

newDestinationProfileRef

A pointer to an opaque struct containing a ColorSync profile.

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

Version Notes
Introduced in QuickTime 6.4.

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

Declared In
`ImageCompression.h`

GraphicsImportSetOverrideSourceColorSyncProfileRef

Sets the override ColorSync profile for a graphics importer component.

```
ComponentResult GraphicsImportSetOverrideSourceColorSyncProfileRef
(
    GraphicsImportComponent   ci,
    CMProfileRef              newOverrideSourceProfileRef);
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

newOverrideSourceProfileRef

A pointer to an opaque struct containing a ColorSync profile.

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

Version Notes
Introduced in QuickTime 6.4.

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

Declared In
`ImageCompression.h`

GraphicsImportWillUseColorMatching

Asks whether `GraphicsImportDraw` will use color matching if called with the current importer settings.

```
ComponentResult GraphicsImportWillUseColorMatching
(
    GraphicsImportComponent   ci,
    Boolean                   *outWillMatch );
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

outWillMatch

On return, a pointer to a Boolean set to `TRUE` if the graphics importer will use color matching, `FALSE` otherwise.

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

Version Notes
Introduced in QuickTime 6.4.

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

Declared In
`ImageCompression.h`

MovieImportDoUserDialogDataRef

Requests that a movie import component display its user dialog box.

```
ComponentResult MovieImportDoUserDialogDataRef (
    MovieImportComponent    ci,
    Handle                  dataRef,
```

```
    OSType                  dataRefType,
    Boolean                 *canceled );
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

dataRef

A data reference that specifies a storage location that contains the data to import.

dataRefType

The type of the data reference.

canceled

A pointer to a `Boolean` entity that is set to `TRUE` if the user cancels the export operation.

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

Discussion
This function brings up the option dialog for the import component. The data reference specified the storage location that contains the data to import.

Version Notes
Introduced in QuickTime 6.4.

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

Declared In
`QuickTimeComponents.h`

MovieImportSetMediaDataRef

Specifies a storage location that is to receive imported movie data.

```
ComponentResult MovieImportSetMediaDataRef (
    MovieImportComponent    ci,
    Handle                  dataRef,
    OSType                  dataRefType );
```

Parameters

ci

The component instance that identifies your connection to the graphics importer component.

dataRef

A data reference that specifies a storage location that receives the imported data.

dataRefType

The type of the data reference.

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

Discussion
By calling this function you can specify a storage location that receives some imported movie data.

Version Notes
Introduced in QuickTime 6.4.

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

Declared In
`QuickTimeComponents.h`

QTAddComponentPropertyListener

Installs a callback to monitor a component property.

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

Parameters

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

inPropClass
A value of type `OSType` that specifies a property class:

`kComponentPropertyClassPropertyInfo ('pnfo')`
A `QTComponentPropertyInfo` (page 89) structure that defines a property information class.

`kComponentPropertyInfoList ('list')`
An array of `QTComponentPropertyInfo` structures, one for each property.

`kComponentPropertyCacheSeed ('seed')`
A component property cache seed value.

`kComponentPropertyExtendedInfo ('meta')`
A `CFDictionary` with extended property information.

`kComponentPropertyCacheFlags ('flgs')`
One of the following two flags:

`kComponentPropertyCacheFlagNotPersistent`
Property metadata should not be saved in persistent cache.

`kComponentPropertyCacheFlagIsDynamic`
Property metadata should not be cached at all.

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

inDispatchProc
A Universal Procedure Pointer to a `QTComponentPropertyListenerProc` (page 88) callback.

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

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

Version Notes
Introduced in QuickTime 6.4.

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

Declared In
`ImageCompression.h`

QTAddMoviePropertyListener

Installs a callback to monitor a movie property.

```
OSErr QTAddMoviePropertyListener (
    Movie                           inMovie,
    QTPropertyClass                 inPropClass,
    QTPropertyID                    inPropID,
    QTMoviePropertyListenerUPP      inListenerProc,
    void                            *inUserData );
```

Parameters

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

inPropClass
A property class.

inPropID
A property ID.

inListenerProc
A Universal Procedure Pointer to a `QTMoviePropertyListenerProc` (page 88) callback.

inUserData
A pointer to user data that will be passed to the callback.

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

Version Notes
Introduced in QuickTime 6.4.

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

Declared In
`Movies.h`

QTComponentPropertyListenerCollectionAddListener

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

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

Parameters

inCollection

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

inPropClass

A value of type `OSType` that specifies a property class:

`kComponentPropertyClassPropertyInfo` (`'pnfo'`)

A `QTComponentPropertyInfo` (page 89) structure that defines a property information class.

`kComponentPropertyInfoList` (`'list'`)

An array of `QTComponentPropertyInfo` structures, one for each property.

`kComponentPropertyCacheSeed` (`'seed'`)

A component property cache seed value.

`kComponentPropertyExtendedInfo` (`'meta'`)

A `CFDictionary` with extended property information.

`kComponentPropertyCacheFlags` (`'flgs'`)

One of the following two flags:

`kComponentPropertyCacheFlagNotPersistent`

Property metadata should not be saved in persistent cache.

`kComponentPropertyCacheFlagIsDynamic`

Property metadata should not be cached at all.

inPropID

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

inListenerProc

A `QTComponentPropertyListenerProc` (page 88) callback.

inListenerProcRefCon

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

Return Value

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

Version Notes

Introduced in QuickTime 6.4.

Availability

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

Declared In

`ImageCompression.h`

QTComponentPropertyListenerCollectionCreate

Creates a collection of component property monitors.

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

Parameters

inAllocator

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

inContext

A pointer to a `QTComponentPropertyInfo` (page 89) data structure. You can pass `NIL` if no structure exists. A copy of the contents of the structure is made; therefore you can pass a pointer to a structure on the stack.

outCollection

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

Return Value

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

Version Notes

Introduced in QuickTime 6.4.

Availability

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

Platform Considerations

This function can be used to create a collection for use with `ComponentPropertyListenerCollectionAddListener`, `ComponentPropertyListenerCollectionRemoveListener`, `ComponentPropertyListenerCollectionNotifyListeners`, `ComponentPropertyListenerCollectionIsEmpty`, and `ComponentPropertyListenerCollectionHasListenersForProperty`.

Declared In

`ImageCompression.h`

QTComponentPropertyListenerCollectionHasListenersForProperty

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

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

Parameters

inCollection

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

inPropClass

A value of type `OSType` that specifies a property class:

`kComponentPropertyClassPropertyInfo ('pnfo')`

A `QTComponentPropertyInfo` (page 89) structure that defines a property information class.

`kComponentPropertyInfoList ('list')`

An array of `QTComponentPropertyInfo` structures, one for each property.

`kComponentPropertyCacheSeed ('seed')`

A component property cache seed value.

`kComponentPropertyExtendedInfo ('meta')`

A `CFDictionary` with extended property information.

`kComponentPropertyCacheFlags ('flgs')`

One of the following two flags:

`kComponentPropertyCacheFlagNotPersistent`

Property metadata should not be saved in persistent cache.

`kComponentPropertyCacheFlagIsDynamic`

Property metadata should not be cached at all.

inPropID

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

Return Value

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

Version Notes

Introduced in QuickTime 6.4.

Availability

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

Declared In

`ImageCompression.h`

QTComponentPropertyListenerCollectionIsEmpty

Determines if a listener collection is empty.

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

Parameters

inCollection

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

Return Value

Returns `TRUE` if the collection is empty, `FALSE` otherwise.

Version Notes

Introduced in QuickTime 6.4.

Availability

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

Declared In

`ImageCompression.h`

QTComponentPropertyListenerCollectionNotifyListeners

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

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

Parameters

inCollection

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

inNotifier

The caller's component instance.

inPropClass

A value of type `OSType` that specifies a property class:

`kComponentPropertyClassPropertyInfo` (`'pnfo'`)

A `QTComponentPropertyInfo` (page 89) structure that defines a property information class.

`kComponentPropertyInfoList` (`'list'`)

An array of `QTComponentPropertyInfo` structures, one for each property.

`kComponentPropertyCacheSeed` (`'seed'`)

A component property cache seed value.

`kComponentPropertyExtendedInfo` (`'meta'`)

A `CFDictionary` with extended property information.

`kComponentPropertyCacheFlags` (`'flgs'`)

One of the following two flags:

`kComponentPropertyCacheFlagNotPersistent`

Property metadata should not be saved in persistent cache.

`kComponentPropertyCacheFlagIsDynamic`

Property metadata should not be cached at all.

inPropID

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

inFilterProcRefCon

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

inFlags

Currently not used.

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

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

Version Notes
Introduced in QuickTime 6.4.

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

Declared In
`ImageCompression.h`

QTComponentPropertyListenerCollectionRemoveListener

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

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

Parameters

inCollection

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

inPropClass

A value of type `OSType` that specifies a property class:

`kComponentPropertyClassPropertyInfo` (`'pnfo'`)

A `QTComponentPropertyInfo` (page 89) structure that defines a property information class.

`kComponentPropertyInfoList` (`'list'`)

An array of `QTComponentPropertyInfo` structures, one for each property.

`kComponentPropertyCacheSeed` (`'seed'`)

A component property cache seed value.

`kComponentPropertyExtendedInfo` (`'meta'`)

A `CFDictionary` with extended property information.

`kComponentPropertyCacheFlags` (`'flgs'`)

One of the following two flags:

`kComponentPropertyCacheFlagNotPersistent`

Property metadata should not be saved in persistent cache.

`kComponentPropertyCacheFlagIsDynamic`

Property metadata should not be cached at all.

inPropID

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

inListenerProc

The `QTComponentPropertyListenerProc` (page 88) callback to be removed.

inListenerProcRefCon

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

Return Value

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

Version Notes

Introduced in QuickTime 6.4.

Availability

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

Declared In
`ImageCompression.h`

QTGetComponentProperty

Returns the value of a specific component property.

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

Parameters

inComponent

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

inPropClass

A value of type `OSType` that specifies a property class:

`kComponentPropertyClassPropertyInfo ('pnfo')`

A `QTComponentPropertyInfo` (page 89) structure that defines a property information class.

`kComponentPropertyInfoList ('list')`

An array of `QTComponentPropertyInfo` structures, one for each property.

`kComponentPropertyCacheSeed ('seed')`

A component property cache seed value.

`kComponentPropertyExtendedInfo ('meta')`

A `CFDictionary` with extended property information.

`kComponentPropertyCacheFlags ('flgs')`

One of the following two flags:

`kComponentPropertyCacheFlagNotPersistent`

Property metadata should not be saved in persistent cache.

`kComponentPropertyCacheFlagIsDynamic`

Property metadata should not be cached at all.

inPropID

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

inPropValueSize

The size of the buffer allocated to hold the property value.

outPropValueAddress

A pointer to the buffer allocated to hold the property value.

outPropValueSizeUsed

On return, the actual size of the value written to the buffer.

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

Version Notes
Introduced in QuickTime 6.4.

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

Declared In
`ImageCompression.h`

QTGetComponentPropertyInfo

Returns information about the properties of a component.

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

Parameters

inComponent

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

inPropClass

A value of type `OSType` that specifies a property class:

`kComponentPropertyClassPropertyInfo` (`'pnfo'`)

A `QTComponentPropertyInfo` (page 89) structure that defines a property information class.

`kComponentPropertyInfoList` (`'list'`)

An array of `QTComponentPropertyInfo` structures, one for each property.

`kComponentPropertyCacheSeed` (`'seed'`)

A component property cache seed value.

`kComponentPropertyExtendedInfo` (`'meta'`)

A `CFDictionary` with extended property information.

`kComponentPropertyCacheFlags` (`'flgs'`)

One of the following two flags:

`kComponentPropertyCacheFlagNotPersistent`

Property metadata should not be saved in persistent cache.

`kComponentPropertyCacheFlagIsDynamic`

Property metadata should not be cached at all.

inPropID

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

outPropType

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

outPropValueSize

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

outPropertyFlags

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

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

Version Notes
Introduced in QuickTime 6.4.

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

Declared In
`ImageCompression.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
Carbon status: Supported; C interface file: `Movies.h`

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 that identifies the syntax of the pathname.

`kQTNativeDefaultPathStyle`

The default pathname syntax of the platform.

`kQTPOSIXPathStyle`

Used on Unix-based systems where pathname components are delimited by slashes.

`kQTHFSPathStyle`

The Macintosh HFS file system syntax where the delimiters are colons.

`kQTWindowsPathStyle`

The Windows pathname syntax that uses backslashes as component delimiters.

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

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

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

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

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     pathStyle,
    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 that identifies the syntax of the pathname.

`kQTNativeDefaultPathStyle`
The default pathname syntax of the platform.

`kQTPOSIXPathStyle`
Used on Unix-based systems where pathname components are delimited by slashes.

`kQTHFSPathStyle`
The Macintosh HFS file system syntax where the delimiters are colons.

`kQTWindowsPathStyle`
The Windows pathname syntax that uses backslashes as component delimiters.

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

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

Declared In
`Movies.h`

QTGetMovieProperty

Returns the value of a specific movie property.

```
OSErr QTGetMovieProperty (
    Movie                   inMovie,
    QTPropertyClass         inPropClass,
    QTPropertyID            inPropID,
    ByteCount               inPropValueSize,
```

```
    QTPropertyValuePtr    outPropValueAddress,
    ByteCount             *outPropValueSizeUsed );
```

Parameters

inMovie

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

inPropClass

A property class.

inPropID

A property ID.

inPropValueSize

The size of the buffer allocated to hold the property value.

outPropValueAddress

A pointer to the buffer allocated to hold the property value.

outPropValueSizeUsed

On return, the actual size of the value written to the buffer.

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

Version Notes
Introduced in QuickTime 6.4.

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

Declared In
`Movies.h`

QTGetMoviePropertyInfo

Returns information about the properties of a movie.

```
OSErr QTGetMoviePropertyInfo (
    Movie                 inMovie,
    QTPropertyClass       inPropClass,
    QTPropertyID          inPropID,
    QTPropertyValueType   *outPropType,
    ByteCount             *outPropValueSize,
    UInt32                *outPropertyFlags );
```

Parameters

inMovie

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

inPropClass

A property class.

inPropID

A property ID.

outPropType
A pointer to memory allocated to hold the property type on return.

outPropValueSize
A pointer to memory allocated to hold the size of the property value on return.

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

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

Version Notes
Introduced in QuickTime 6.4.

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

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

Declared In
Movies.h

QTNewDataReferenceFromFSRef

Creates an alias data reference from a file specification.

```
OSErr QTNewDataReferenceFromFSRef (
    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 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
Carbon status: Supported; C interface file: Movies.h

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

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

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

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 that identifies the syntax of the pathname.

`kQTNativeDefaultPathStyle`
The default pathname syntax of the platform.

`kQTPOSIXPathStyle`
Used on Unix-based systems where pathname components are delimited by slashes.

`kQTHFSPathStyle`
The Macintosh HFS file system syntax where the delimiters are colons.

`kQTWindowsPathStyle`
The Windows pathname syntax that uses backslashes as component delimiters.

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

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

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

Declared In
Movies.h

QTRemoveComponentPropertyListener

Removes a component property monitoring callback.

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

Parameters

inComponent

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

inPropClass

A value of type `OSType` that specifies a property class:

`kComponentPropertyClassPropertyInfo` (`'pnfo'`)

A `QTComponentPropertyInfo` (page 89) structure that defines a property information class.

`kComponentPropertyInfoList` (`'list'`)

An array of `QTComponentPropertyInfo` structures, one for each property.

`kComponentPropertyCacheSeed` (`'seed'`)

A component property cache seed value.

`kComponentPropertyExtendedInfo` (`'meta'`)

A `CFDictionary` with extended property information.

`kComponentPropertyCacheFlags` (`'flgs'`)

One of the following two flags:

`kComponentPropertyCacheFlagNotPersistent`

Property metadata should not be saved in persistent cache.

`kComponentPropertyCacheFlagIsDynamic`

Property metadata should not be cached at all.

inPropID

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

inDispatchProc

A Universal Procedure Pointer to a `QTComponentPropertyListenerProc` (page 88) callback.

inUserData

User data to be passed to the callback.

Return Value

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

Version Notes

Introduced in QuickTime 6.4.

Availability

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

Declared In

`ImageCompression.h`

QTRemoveMoviePropertyListener

Removes a movie property monitoring callback.

```
OSErr QTRemoveMoviePropertyListener (
    Movie                       inMovie,
    QTPropertyClass             inPropClass,
    QTPropertyID                inPropID,
    QTMoviePropertyListenerUPP  inListenerProc,
    void                        *inUserData );
```

Parameters

inMovie

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

inPropClass

A property class.

inPropID

A property ID.

inListenerProc

A Universal Procedure Pointer to a `QTMoviePropertyListenerProc` (page 88) callback.

inUserData

User data to be passed to the callback.

Return Value

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

Version Notes

Introduced in QuickTime 6.4.

Availability

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

Declared In

`Movies.h`

QTSetComponentProperty

Sets the value of a specific component property.

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

Parameters

inComponent

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

inPropClass
A value of type `OSType` that specifies a property class:

`kComponentPropertyClassPropertyInfo` (`'pnfo'`)
A `QTComponentPropertyInfo` (page 89) structure that defines a property information class.

`kComponentPropertyInfoList` (`'list'`)
An array of `QTComponentPropertyInfo` structures, one for each property.

`kComponentPropertyCacheSeed` (`'seed'`)
A component property cache seed value.

`kComponentPropertyExtendedInfo` (`'meta'`)
A `CFDictionary` with extended property information.

`kComponentPropertyCacheFlags` (`'flgs'`)
One of the following two flags:

`kComponentPropertyCacheFlagNotPersistent`
Property metadata should not be saved in persistent cache.

`kComponentPropertyCacheFlagIsDynamic`
Property metadata should not be cached at all.

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

inPropValueSize
The size of the buffer allocated to hold the property value.

outPropValueAddress
A pointer to the buffer allocated to hold the property value.

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

Version Notes
Introduced in QuickTime 6.4.

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

Declared In
`ImageCompression.h`

QTSetMovieProperty

Sets the value of a specific movie property.

```
OSErr QTSetMovieProperty (
    Movie                       inMovie,
    QTPropertyClass             inPropClass,
    QTPropertyID                inPropID,
    ByteCount                   inPropValueSize,
    ConstQTPropertyValuePtr     inPropValueAddress );
```

Parameters

inMovie

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

inPropClass

A property class.

inPropID

A property ID.

inPropValueSize

The size of the buffer allocated to hold the property value.

outPropValueAddress

A pointer to the buffer allocated to hold the property value.

Return Value

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

Version Notes

Introduced in QuickTime 6.4.

Availability

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

Declared In

`Movies.h`

QTVideoOutputCopyIndAudioOutputDeviceUID

Identifies the audio device being used by a video output component.

```
OSErr QTVideoOutputCopyIndAudioOutputDeviceUID (
   QTVideoOutputComponent    vo,
   long                      index,
   CFStringRef               *audioDeviceUID );
```

Parameters

vo

Video output component whose audio output is being asked about.

index

Which of video output component's audio outputs is being asked about.

audioDeviceUID

Returned unique identifier for the audio device. If the UID is `NIL`, the movie is playing to the default device.

Return Value

See "Error Codes" in the QuickTime API Reference. Returns `noErr` if there is no error. Returns `badComponentInstance` if `vo` is not a valid ComponentInstance. Returns `badComponentSelector` if `vo` doesn't support this function. Returns `paramErr` if `audioDeviceUID` is `NIL`, or if there is no device with the passed index.

Discussion

The returned `audioDeviceUID` has already been retained for the caller, using standard Core Foundation copy semantics.

Version Notes
Introduced in QuickTime 6.4.

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

Declared In
`QuickTimeComponents.h`

SetTimeBaseOffsetTimeBase

Attaches an offset time base to another time base.

```
OSErr SetTimeBaseOffsetTimeBase (
    TimeBase            masterOffsetTimeBase,
    TimeBase            offsetTimeBase,
    const TimeRecord    *offsetZero );
```

Parameters

`masterOffsetTimeBase`
The time base to which the offset time base is to be attached. A `NIL` value can be passed when the offset time base has already be set but a new offset value is needed.

`offsetTimeBase`
The offset time base to be attached.

`offsetZero`
A pointer to a `TimeRecord` value set to the offset between the master time base and the offset time base. Passing a negative value means the offset time base will start sooner.

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

Version Notes
Introduced in QuickTime 6.4.

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

Declared In
`Movies.h`

VDIIDCGetCSRData

Reads a camera's CSR registers directly.

```
VideoDigitizerError VDIIDCGetCSRData (
    VideoDigitizerComponent    ci,
    Boolean                    offsetFromUnitBase,
    UInt32                     offset,
    UInt32*                    data );
```

Parameters

ci

The component instance that identifies your connection to a video digitizer component. The digitizer's subtype must be `vdSubtypeIIDC ('iidc')`.

offsetFromUnitBase

Pass `TRUE` if the offset is relative to the initial unit space (`FFFF Fxxx xxxx`), `FALSE` if the offset is relative to the initial register space (`FFFF F000 0000`).

offset

Offset in bytes of the value to read.

data

Location to store the value (of type `UInt32`) that was read.

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

Discussion
You might want to read a camera's registers directly if you're querying the state of a feature not accessed by `VDIIDCGetFeatures` or if some camera-specific information must be accessed.

Special Considerations

The initial release of the QuickTime IIDC Digitizer improperly used a value of `FFFF 0000 0000` for the start of the register space. If using the QuickTime IIDC Digitizer and `0x00020100 == GetComponentVersion()`, you must add `0xF000 0000` to the offset when accessing the register space.

Version Notes
Introduced in QuickTime 6.4.

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

Declared In
`QuickTimeComponents.h`

VDIIDCGetDefaultFeatures

Places atoms in a QuickTime atom container that specify the default capabilities and default state of a camera's IIDC features.

```
VideoDigitizerError VDIIDCGetDefaultFeatures (
    VideoDigitizerComponent    ci,
    QTAtomContainer            *container );
```

Parameters

ci

The component instance that identifies your connection to a video digitizer component. The digitizer's subtype must be `vdSubtypeIIDC ('iidc')`.

container

Upon return, a pointer to a QuickTime atom container containing atoms of type `vdIIDCAtomTypeFeature` for each IIDC camera feature whose default is known. See "IIDC Feature Atoms" (page 25). The container may be empty if defaults cannot be determined.

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

Discussion
The digitizer will create the QuickTime atom container, and it is the responsibility of the client to delete it if the routine does not return an error.

Version Notes
Introduced in QuickTime 6.4.

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

Declared In
`QuickTimeComponents.h`

VDIIDCGetFeatures

Places atoms in a QuickTime atom container that specify the current capabilities of a camera and the state of its IIDC features.

```
VideoDigitizerError VDIIDCGetFeatures (
    VideoDigitizerComponent    ci,
    QTAtomContainer            *container );
```

Parameters

ci

The component instance that identifies your connection to a video digitizer component. The digitizer's subtype must be `vdSubtypeIIDC ('iidc')`.

container

Upon return, a pointer to a QuickTime atom container containing atoms of type `vdIIDCAtomTypeFeature` for each IIDC camera feature. See "IIDC Feature Atoms" (page 25). If the camera has not implemented any IIDC features the container returns empty.

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

Discussion
The digitizer creates the container, and it is the responsibility of the client to ultimately delete it if the routine does not return an error. Since the values that this function retrieves might change underneath the client, they should not be cached but should be retrieved each time they are needed.

Version Notes
Introduced in QuickTime 6.4.

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

Declared In
`QuickTimeComponents.h`

VDIIDCGetFeaturesForSpecifier

Places atoms in a QuickTime atom container that specify the current state of a single camera IIDC feature or group of features.

```
VideoDigitizerError VDIIDCGetFeaturesForSpecifier (
    VideoDigitizerComponent    ci,
    OSType                     specifier,
    QTAtomContainer            *container );
```

Parameters

ci

The component instance that identifies your connection to a video digitizer component. The digitizer's subtype must be `vdSubtypeIIDC ('iidc')`.

specifier

The feature or group of features to be retrieved:

```
// IIDC feature types
    vdIIDCFeatureHue                = 'hue ',
    vdIIDCFeatureSaturation         = 'satu',
    vdIIDCFeatureSharpness          = 'shrp',
    vdIIDCFeatureBrightness         = 'brit',
    vdIIDCFeatureGain               = 'gain',
    vdIIDCFeatureIris               = 'iris',
    vdIIDCFeatureShutter            = 'shtr',
    vdIIDCFeatureExposure           = 'xpsr',
    vdIIDCFeatureWhiteBalanceU      = 'whbu',
    vdIIDCFeatureWhiteBalanceV      = 'whbv',
    vdIIDCFeatureGamma              = 'gmma',
    vdIIDCFeatureTemperature        = 'temp',
    vdIIDCFeatureZoom               = 'zoom',
    vdIIDCFeatureFocus              = 'fcus',
    vdIIDCFeaturePan                = 'pan ',
    vdIIDCFeatureTilt               = 'tilt',
    vdIIDCFeatureOpticalFilter      = 'opft',
    vdIIDCFeatureTrigger            = 'trgr',
    vdIIDCFeatureCaptureSize        = 'cpsz',
    vdIIDCFeatureCaptureQuality     = 'cpql',
    vdIIDCFeatureFocusPoint         = 'fpnt',
    vdIIDCFeatureEdgeEnhancement    = 'eden'
    vdIIDCFeatureLightingHint       = 'lhnt'

// IIDC group types
    vdIIDCGroupImage                = 'imag',
    vdIIDCGroupColor                = 'colr',
    vdIIDCGroupMechanics            = 'mech',
    vdIIDCGroupTrigger              = 'trig'
```

container

Upon return, a pointer to a QuickTime atom container containing atoms of type `vdIIDCAtomTypeFeature` for each IIDC camera feature corresponding to the specifier. See "IIDC Feature Atoms" (page 25). If the camera has not implemented any of the specified features the container returns empty.

Return Value

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

Discussion

The digitizer creates the container, and it is the responsibility of the client to ultimately delete it if the routine does not return an error. Since the values that this function retrieves might change underneath the client, they should not be cached but should be retrieved each time they are needed.

Version Notes
Introduced in QuickTime 6.4.

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

Declared In
`QuickTimeComponents.h`

VDIIDCSetCSRData

Writes to a camera's CSR registers directly.

```
VideoDigitizerError VDIIDCSetCSRData (
    VideoDigitizerComponent    ci,
    Boolean                    offsetFromUnitBase,
    UInt32                     offset,
    UInt32*                    data );
```

Parameters

ci

The component instance that identifies your connection to a video digitizer component. The digitizer's subtype must be `vdSubtypeIIDC ('iidc')`.

offsetFromUnitBase

Pass `TRUE` if the offset is relative to the initial unit space (`FFFF Fxxx xxxx`), `FALSE` if the offset is relative to the initial register space (`FFFF F000 0000`).

offset

Offset in bytes of the value to set.

data

Location of the value (of type `UInt32`) to write.

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

Discussion
You might want to write to a camera's registers directly if you're setting the state of a feature not accessed by `VDIIDCSetFeatures` or if some camera-specific information must be set.

Special Considerations

The initial release of the QuickTime IIDC Digitizer improperly used a value of `FFFF 0000 0000` for the start of the register space. If using the QuickTime IIDC Digitizer and `0x00020100 == GetComponentVersion()`, you must add `0xF000 0000` to the offset when accessing the register space.

Version Notes
Introduced in QuickTime 6.4.

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

Declared In
`QuickTimeComponents.h`

VDIIDCSetFeatures

Changes the state of a camera's IIDC features.

```
VideoDigitizerError VDIIDCSetFeatures (
    VideoDigitizerComponent    ci,
    QTAtomContainer            *container );
```

Parameters

ci

The component instance that identifies your connection to a video digitizer component. The digitizer's subtype must be `vdSubtypeIIDC ('iidc')`.

container

A pointer to a QuickTime atom container populated with atoms of type `vdIIDCAtomTypeFeature`; see "IIDC Feature Atoms" (page 25). The container may have one or many atoms in it. An empty container will cause the function to have no effect.

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

Discussion
It is the responsibility of the client to provide the QuickTime atom container and delete it after use.

Version Notes
Introduced in QuickTime 6.4.

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

Declared In
`QuickTimeComponents.h`

Callbacks

The callback functions new to the QuickTime 6.4 API are documented alphabetically in this section.

QTComponentPropertyListenerFilterProc

Supports `QTComponentPropertyListenerCollectionNotifyListeners` (page 60).

```
typedef Boolean (*QTComponentPropertyListenerFilterProcPtr)
(QTComponentPropertyListenersRef
inCollection, const QTComponentPropertyListenerCollectionContext
*inCollectionContext, ComponentInstance inNotifier, ComponentPropertyClass
inPropClass, ComponentPropertyID inPropID, QTComponentPropertyListenerUPP
inListenerCallbackProc,
const void *inListenerProcRefCon, const void *inFilterProcRefCon);

// Declaration of a typical application-defined function
Boolean MyQTComponentPropertyListenerFilterProc (
    QTComponentPropertyListenersRef    inCollection,
    const QTComponentPropertyListenerCollectionContext
                                       *inCollectionContext,
    ComponentInstance                  inNotifier,
```

```
    ComponentPropertyClass              inPropClass,
    ComponentPropertyID                 inPropID,
    QTComponentPropertyListenerUPP      inListenerCallbackProc,
    const void                          *inListenerProcRefCon,
    const void                          *inFilterProcRefCon );
```

Parameters

inCollection

The property listener reference that was returned by a previous call to `QTComponentPropertyListenerCollectionCreate`.

inCollectionContext

A pointer to a `QTComponentPropertyInfo` (page 89) structure.

inNotifier

An instance of the notifying component.

inPropClass

A value of type `OSType` that specifies a property class:

`kComponentPropertyClassPropertyInfo` (`'pnfo'`)

A `QTComponentPropertyInfo` (page 89) structure that defines a property information class.

`kComponentPropertyInfoList` (`'list'`)

An array of `QTComponentPropertyInfo` structures, one for each property.

`kComponentPropertyCacheSeed` (`'seed'`)

A component property cache seed value.

`kComponentPropertyExtendedInfo` (`'meta'`)

A `CFDictionary` with extended property information.

`kComponentPropertyCacheFlags` (`'flgs'`)

One of the following two flags:

`kComponentPropertyCacheFlagNotPersistent`

Property metadata should not be saved in persistent cache.

`kComponentPropertyCacheFlagIsDynamic`

Property metadata should not be cached at all.

inPropID

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

inListenerCallbackProc

A `QTComponentPropertyListenerProc` (page 88) callback.

inListenerProcRefCon

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

inFilterProcRefCon

A reference constant to be passed to the callback specified in the `filterProcUPP` field of the `QTComponentPropertyListenerCollectionContext` structure pointed to by the `inCollectionContext` parameter. Use this parameter to point to a data structure containing any information your callback needs.

QTComponentPropertyListenerProc

Supports QTAddComponentPropertyListener (page 55) and QTRemoveComponentPropertyListener (page 76).

```
typedef void (*QTComponentPropertyListenerProcPtr)
(ComponentInstance inComponent, ComponentPropertyClass inPropClass,
ComponentPropertyID inPropID, void *inUserData);

// Declaration of a typical application-defined function
void MyQTComponentPropertyListenerProc (
    ComponentInstance         inComponent,
    ComponentPropertyClass    inPropClass,
    ComponentPropertyID       inPropID,
    void                      *inUserData );
```

Parameters

inComponent

A reference to the component for this operation.

inPropClass

A value of type OSType that specifies a property class:

kComponentPropertyClassPropertyInfo ('pnfo')

A QTComponentPropertyInfo (page 89) structure that defines a property information class.

kComponentPropertyInfoList ('list')

An array of QTComponentPropertyInfo structures, one for each property.

kComponentPropertyCacheSeed ('seed')

A component property cache seed value.

kComponentPropertyExtendedInfo ('meta')

A CFDictionary with extended property information.

kComponentPropertyCacheFlags ('flgs')

One of the following two flags:

kComponentPropertyCacheFlagNotPersistent

Property metadata should not be saved in persistent cache.

kComponentPropertyCacheFlagIsDynamic

Property metadata should not be cached at all.

inPropID

A value of type OSType that specifies a property ID.

inUserData

A pointer to data that QuickTime will pass to your callback.

QTMoviePropertyListenerProc

Supports QTAddMoviePropertyListener (page 56) and QTRemoveMoviePropertyListener (page 78).

```
typedef void (*QTMoviePropertyListenerProcPtr) (Movie
inMovie, QTPropertyClass inPropClass, QTPropertyID inPropID, void
*inUserData);
```

```
// Declaration of a typical application-defined function
void MyQTMoviePropertyListenerProc (
    Movie              inMovie,
    QTPropertyClass    inPropClass,
    QTPropertyID       inPropID,
    void               *inUserData );
```

Parameters

inMovie

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

inPropClass

A property class.

inPropID

A property ID.

inUserData

A pointer to data for your callback.

Data Structures

The data structures new to the QuickTime 6.4 API are documented alphabetically in this section.

QTComponentPropertyInfo

Stores information for component property functions.

```
struct ComponentPropertyInfo {
    ComponentPropertyClass    propClass;
    ComponentPropertyID       propID;
    ComponentValueType        propType;
    ByteCount                 propSize;
    UInt32                    propFlags;
};
```

Fields

`propClass`

A value of type `OSType` that specifies a property class:

`kComponentPropertyClassPropertyInfo ('pnfo')`

A `QTComponentPropertyInfo` (page 89) structure that defines a property information class.

`kComponentPropertyInfoList ('list')`

An array of `QTComponentPropertyInfo` structures, one for each property.

`kComponentPropertyCacheSeed ('seed')`

A component property cache seed value.

`kComponentPropertyExtendedInfo ('meta')`

A `CFDictionary` with extended property information.

`kComponentPropertyCacheFlags ('flgs')`

One of the following two flags:

`kComponentPropertyCacheFlagNotPersistent`

Property metadata should not be saved in persistent cache.

`kComponentPropertyCacheFlagIsDynamic`

Property metadata should not be cached at all.

`propID`

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

`propType`

The type of the property.

`propSize`

The size of the property in bytes.

`propFlags`

Component property flags.

Version Notes

Introduced in QuickTime 6.4.

Availability

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

Platform Considerations

Associated functions: `QTAddComponentPropertyListener`, `QTComponentPropertyListenerCollectionAddListener`, `QTComponentPropertyListenerCollectionHasListenersForProperty`, `QTComponentPropertyListenerCollectionNotifyListeners`, `QTComponentPropertyListenerCollectionRemoveListener`, `QTGetComponentProperty`, `QTGetComponentPropertyInfo`, `QTRemoveComponentPropertyListener`, `QTSetComponentProperty`, `QTComponentPropertyListenerFilterProc`, `QTComponentPropertyListenerProc`

QTComponentPropertyListenerCollectionContext

Provides context information for a `QTComponentPropertyListenerFilterProc` callback.

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

Fields

`version`
The version of this callback.

`filterProcUPP`
A Universal Procedure Pointer to a `QTComponentPropertyListenerFilterProc` (page 86) callback.

`filterProcData`
A pointer to data for the callback.

Version Notes
Introduced in QuickTime 6.4.

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

Platform Considerations

Associated function: `QTComponentPropertyListenerCollectionNotifyListeners`

VDIIDCFeatureAtomTypeAndID

Provides content for the `vdIIDCAtomTypeFeatureAtomTypeAndID` atom type. See "Type And ID Atoms" (page 25).

```
struct VDIIDCFeatureAtomTypeAndID {
    OSType        feature;
    OSType        group;
    Str255        name;
    QTAtomType    atomType;
    QTAtomID      atomID;
};
```

Fields

`feature`
Type of the feature (see Discussion below).

`group`
Group that categorizes the feature (see Discussion below).

`name`
Name of the feature.

`atomType`
Atom type that contains the feature's settings. See "IIDC Settings Atoms" (page 25).

`atomID`
Atom ID of the atom that contains the feature's settings.

Discussion
The following are the possible values for the `feature` and `group` parameters:

```
// IIDC Feature OSTypes
    vdIIDCFeatureHue            = 'hue ', // Uses VDIIDCFeatureSettings
    vdIIDCFeatureSaturation     = 'satu', // Uses VDIIDCFeatureSettings
    vdIIDCFeatureSharpness      = 'shrp', // Uses VDIIDCFeatureSettings
    vdIIDCFeatureBrightness     = 'brit', // Uses VDIIDCFeatureSettings
    vdIIDCFeatureGain           = 'gain', // Uses VDIIDCFeatureSettings
    vdIIDCFeatureIris           = 'iris', // Uses VDIIDCFeatureSettings
    vdIIDCFeatureShutter        = 'shtr', // Uses VDIIDCFeatureSettings
    vdIIDCFeatureExposure       = 'xpsr', // Uses VDIIDCFeatureSettings
    vdIIDCFeatureWhiteBalanceU  = 'whbu', // Uses VDIIDCFeatureSettings
    vdIIDCFeatureWhiteBalanceV  = 'whbv', // Uses VDIIDCFeatureSettings
    vdIIDCFeatureGamma          = 'gmma', // Uses VDIIDCFeatureSettings
    vdIIDCFeatureTemperature    = 'temp', // Uses VDIIDCFeatureSettings
    vdIIDCFeatureZoom           = 'zoom', // Uses VDIIDCFeatureSettings
    vdIIDCFeatureFocus          = 'fcus', // Uses VDIIDCFeatureSettings
    vdIIDCFeaturePan            = 'pan ', // Uses VDIIDCFeatureSettings
    vdIIDCFeatureTilt           = 'tilt', // Uses VDIIDCFeatureSettings
    vdIIDCFeatureOpticalFilter  = 'opft', // Uses VDIIDCFeatureSettings
    vdIIDCFeatureTrigger        = 'trgr', // Uses VDIIDCTriggerSettings
    vdIIDCFeatureCaptureSize    = 'cpsz', // Undefined settings
    vdIIDCFeatureCaptureQuality = 'cpql', // Undefined settings
    vdIIDCFeatureFocusPoint     = 'fpnt', // Use VDIIDCFocusPointSettings
    vdIIDCFeatureEdgeEnhancement = 'eden' // Uses VDIIDCFeatureSettings
    vdIIDCFeatureLightingHint   = 'lhnt'  // Use VDIIDCLightingHintSettings

// IIDC Group OSTypes that features are categorized into
    vdIIDCGroupImage        = 'imag',    // Related to image control
    vdIIDCGroupColor        = 'colr',    // Related to color control
    vdIIDCGroupMechanics    = 'mech',    // Related to mechanics
    vdIIDCGroupTrigger      = 'trig'     // Related to trigger
```

Version Notes
Introduced in QuickTime 6.4.

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

Platform Considerations

Associated functions: `VDIIDCGetDefaultFeatures`, `VDIIDCGetFeatures`, `VDIIDCGetFeaturesForSpecifier`

VDIIDCFeatureCapabilities

Provides IIDC feature capabilities information to the `VDIIDCFeatureSettings` (page 93) data structure.

```
struct VDIIDCFeatureCapabilities {
    UInt32          flags;
    UInt16          rawMinimum;
    UInt16          rawMaximum;
    QTFloatSingle   absoluteMinimum;
    QTFloatSingle   absoluteMaximum;
};
```

Fields

`flags`
Flags that determine characteristics of a given feature.

`vdIIDCFeatureFlagOn`
Set if feature can be turned on.

`vdIIDCFeatureFlagOff`
Set if the feature can be turned off.

`vdIIDCFeatureFlagManual`
Set if the feature can be put into manual mode.

`vdIIDCFeatureFlagAuto`
Set if the feature can be put into automatic mode.

`vdIIDCFeatureFlagTune`
Set if the feature can be tuned. When tuned, a feature drops into automatic mode until it stabilizes and then it reverts to manual mode.

`vdIIDCFeatureFlagRawControl`
Set if the feature's value can be specified in raw values. Raw values are unitless and their meaning can vary from camera to camera.

`vdIIDCFeatureFlagAbsoluteControl`
Set if the feature's value can be specified in absolute units. Absolute values are expressed in engineering units, such as dB or degrees.

`rawMinimum`
Raw minimum value.

`rawMaximum`
Raw maximum value.

`absoluteMinimum`
Absolute minimum value.

`absoluteMaximum`
Absolute maximum value.

Version Notes
Introduced in QuickTime 6.4.

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

Platform Considerations

Associated functions: `VDIIDCGetDefaultFeatures`, `VDIIDCGetFeatures`, `VDIIDCGetFeaturesForSpecifier`

VDIIDCFeatureSettings

Provides content for the `vdIIDCAtomTypeFeatureSettings` atom type. See "IIDC Feature Atoms" (page 25).

```
struct VDIIDCFeatureSettings {
    VDIIDCFeatureCapabilities    capabilities;
    VDIIDCFeatureState           state;
};
```

Fields

`capabilities`

A `VDIIDCFeatureCapabilities` (page 92) data structure that describes a camera's feature capabilities.

`state`

A `VDIIDCFeatureState` (page 94) data structure that describes a camera's current feature state.

Discussion

This data structure can be used to hold data for the following IIDC features: `vdIIDCFeatureHue`, `vdIIDCFeatureSaturation`, `vdIIDCFeatureSharpness`, `vdIIDCFeatureBrightness`, `vdIIDCFeatureGain`, `vdIIDCFeatureIris`. `vdIIDCFeatureShutter`, `vdIIDCFeatureExposure`, `vdIIDCFeatureWhiteBalanceU`, `vdIIDCFeatureWhiteBalanceV`, `vdIIDCFeatureGamma`, `vdIIDCFeatureTemperature`, `vdIIDCFeatureZoom`, `vdIIDCFeatureFocus`, `vdIIDCFeaturePan`, `vdIIDCFeatureTilt`, `vdIIDCFeatureOpticalFilter`, `vdIIDCFeatureEdgeEnhancment`

Version Notes

Introduced in QuickTime 6.4.

Availability

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

Platform Considerations

Associated functions: `VDIIDCGetDefaultFeatures`, `VDIIDCGetFeatures`, `VDIIDCGetFeaturesForSpecifier`, `VDIIDCSetFeatures` (uses only the `state` field)

VDIIDCFeatureState

Provides IIDC feature state information for the `VDIIDCFeatureSettings` (page 93) data structure.

```
struct VDIIDCFeatureState {
    UInt32          flags;
    QTFloatSingle   value;
};
```

Fields

`flags`

Feature flags.

`value`

Feature value.

Version Notes

Introduced in QuickTime 6.4.

Availability

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

Platform Considerations

Associated functions: `VDIIDCGetDefaultFeatures`, `VDIIDCGetFeatures`, `VDIIDCGetFeaturesForSpecifier`, `VDIIDCSetFeatures`

VDIIDCFocusPointSettings

Provides focus point data for the `vdIIDCAtomTypeFocusPointSettings` atom type. See "IIDC Settings Atoms" (page 25).

```
struct VDIIDCFocusPointSettings {
    Point    focusPoint;
};
```

Fields

`focusPoint`

A focus point value relative to the rectangle returned by `SGGetSrcVideoBounds`. When using this data structure to set the focus point, the focus point must be within the rectangle specified by `SGSetVideoRect`.

Version Notes

Introduced in QuickTime 6.4.

Availability

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

Platform Considerations

Associated functions: `VDIIDCGetDefaultFeatures`, `VDIIDCGetFeatures`, `VDIIDCGetFeaturesForSpecifier`, `VDIIDCSetFeatures`

Special Considerations

The focus point feature is not part of the IIDC 1394-based Digital Camera Specification.

VDIIDCLightingHintSettings

Provides lighting hint data for the `vdIIDCAtomTypeLightingHintSettings` atom type. See "IIDC Settings Atoms" (page 25).

```
struct VDIIDCLightingHintSettings {
    UInt32    capabilityFlags;
    UInt32    stateFlags;
};
```

Fields

`capabilityFlags`

Flags that specify particular lighting hint capabilities. This field is ignored by the `VDIIDCSetFeatures` function:

`vdIIDCLightingHintNormal`

Set if the camera supports the normal light hint.

`vdIIDCLightingHintLow`

Set if the camera supports the low light hint.

`stateFlags`

Flags that specify particular lighting hint states:

`vdIIDCLightingHintNormal`

Set if the camera is using or should be using the normal light hint.

`vdIIDCLightingHintLow`

Set if the camera is using or should be using the low light hint.

Version Notes

Introduced in QuickTime 6.4.

Availability

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

Platform Considerations

Associated functions: `VDIIDCGetDefaultFeatures`, `VDIIDCGetFeatures`, `VDIIDCGetFeaturesForSpecifier`, `VDIIDCSetFeatures`

Special Considerations

The lighting hint feature is not part of the IIDC 1394-based Digital Camera Specification.

VDIIDCTriggerCapabilities

Provides IIDC trigger capability information for the `VDIIDCTriggerSettings` (page 98) data structure.

```
struct VDIIDCTriggerCapabilities {
  UInt32           flags;
  QTFloatSingle    absoluteMinimum;
  QTFloatSingle    absoluteMaximum;
};
```

Fields

`flags`

Flags that specify particular trigger capabilities:

`vdIIDCTriggerFlagOn`

Set if the trigger can be turned on.

`vdIIDCTriggerFlagOff`

Set if the trigger can be turned off.

`vdIIDCTriggerFlagActiveHigh`

Set if the trigger can be active high.

`vdIIDCTriggerFlagActiveLow`

Set if the trigger can be active low.

`vdIIDCTriggerFlagMode0`

Set if the trigger can operate in mode 0. In mode 0, the camera starts integrating light at the active edge of the external trigger. The integration time is controlled by the shutter feature.

`vdIIDCTriggerFlagMode1`

Set if the trigger can operate in mode 1. In mode 1, the camera starts integrating light at the active edge of the external trigger. The integration time is equal to the time the trigger is active.

`vdIIDCTriggerFlagMode2`

Set if the trigger can operate in mode 2. In mode 2, the camera starts integrating light at the active edge of the external trigger. Integration continues until the nth active edge, where n is greater than or equal to 2.

`vdIIDCTriggerFlagMode3`

Set if the trigger can operate in mode 3. In mode 3, the camera starts integrating light at the active edge of the internal trigger. The trigger's cycle time is n times the cycle time of the fastest frame rate, where n is greater than or equal to 1. Integration time is controlled by the shutter feature.

`vdIIDCTriggerFlagRawControl`

Set if the trigger's value can be specified in unitless terms.

`vdIIDCTriggerFlagAbsoluteControl`

Set if the trigger's value can be specified in absolute (engineering) units of time.

`absoluteMinimum`

The minimum trigger value when `vdIIDCTriggerFlagAbsoluteControl` is set.

`absoluteMaximum`

The maximum trigger value when `vdIIDCTriggerFlagAbsoluteControl` is set.

Discussion

This data structure is ignored by the `VDIIDCSetFeatures` (page 86) function.

Version Notes

Introduced in QuickTime 6.4.

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

Platform Considerations

Associated functions: `VDIIDCGetDefaultFeatures`, `VDIIDCGetFeatures`, `VDIIDCGetFeaturesForSpecifier`,

VDIIDCTriggerSettings

Provides trigger data for the `vdIIDCAtomTypetriggerSettings` atom type. See "IIDC Settings Atoms" (page 25).

```
struct VDIIDCTriggerSettings {
    VDIIDCTriggerCapabilities    capabilities;
    VDIIDCTriggerState           state;
};
```

Fields

`capabilities`
A `VDIIDCTriggerCapabilities` (page 96) data structure.

`state`
A `VDIIDCTriggerState` (page 98) data structure.

Version Notes
Introduced in QuickTime 6.4.

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

Platform Considerations

Associated functions: `VDIIDCGetDefaultFeatures`, `VDIIDCGetFeatures`, `VDIIDCGetFeaturesForSpecifier`, `VDIIDCSetFeatures`

VDIIDCTriggerState

Provides IIDC trigger state information for the `VDIIDCTriggerSettings` (page 98) data structure.

```
struct VDIIDCTriggerState {
    UInt32          flags;
    UInt16          mode2TransitionCount;
    UInt16          mode3FrameRateMultiplier;
    QTFloatSingle   absoluteValue;
};
```

Fields

`flags`

Flags that specify current or default trigger settings (depending on whether `VDIIDCGetFeatures`, `VDIIDCSetFeatures`, or `VDIIDCGetDefaultFeatures` was originally called):

`vdIIDCTriggerFlagOn`

Set if the trigger is or should be turned on.

`vdIIDCTriggerFlagOff`

Set if the trigger is or should be turned off.

`vdIIDCTriggerFlagActiveHigh`

Set if the trigger is or should be active high.

`vdIIDCTriggerFlagActiveLow`

Set if the trigger is or should be active low.

`vdIIDCTriggerFlagMode0`

Set if the trigger is or should operate in mode 0. In mode 0, the camera starts integrating light at the active edge of the external trigger. The integration time is controlled by the shutter feature.

`vdIIDCTriggerFlagMode1`

Set if the trigger is or should operate in mode 1. In mode 1, the camera starts integrating light at the active edge of the external trigger. The integration time is equal to the time the trigger is active.

`vdIIDCTriggerFlagMode2`

Set if the trigger is or should operate in mode 2. In mode 2, the camera starts integrating light at the active edge of the external trigger. Integration continues until the nth active edge, where n is greater than or equal to 2.

`vdIIDCTriggerFlagMode3`

Set if the trigger is or should operate in mode 3. In mode 3, the camera starts integrating light at the active edge of the internal trigger. The trigger's cycle time is n times the cycle time of the fastest frame rate, where n is greater than or equal to 1. Integration time is controlled by the shutter feature.

`vdIIDCTriggerFlagRawControl`

Set if the trigger's value is or should be specified in unitless terms.

`vdIIDCTriggerFlagAbsoluteControl`

Set if the trigger's value is or should be specified in absolute (engineering) units of time.

`mode2TransitionCount`

When mode 2 is set, the current or default number of transitions.

`mode3FrameRateMultiplier`

When mode 3 is set, the current or default frame rate multiplier.

`absoluteValue`

When using absolute units, the `QTFloatSingle` representation of the value in either `mode2TransitionCount` (if mode 2 is set) or `mode3FrameRateMultiplier` (if mode 3 is set).

Version Notes
Introduced in QuickTime 6.4.

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

Platform Considerations

Associated functions: `VDIIDCGetDefaultFeatures`, `VDIIDCGetFeatures`, `VDIIDCGetFeaturesForSpecifier`, `VDIIDCSetFeatures`

VDIIDC Call Selectors

The following selectors apply to VDIIDC component calls:

```
kVDIIDCGetFeaturesSelect               = 0x0200,
kVDIIDCSetFeaturesSelect               = 0x0201,
kVDIIDCGetDefaultFeaturesSelect        = 0x0202,
kVDIIDCGetCSRDataSelect                = 0x0203,
kVDIIDCSetCSRDataSelect                = 0x0204,
kVDIIDCGetFeaturesForSpecifierSelect   = 0x0205
```