QuickTime Movie Internals Guide

QuickTime > Movie Internals

2006-01-10

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

Simultaneously published in the United States and Canada.

Contents

Figures, Tables, and Listings

Introduction to QuickTime Movie Internals Guide

This book covers some of the technology present inside QuickTime movies, including time management, modifier tracks, access keys, posters, and movie and file previews.

Note: This book replaces five previously separate Apple documents: "Movie Toolbox: Time and Spatial Characteristics," "Clock Components," "Movie Toolbox: Previews," "Movie Toolbox: Access Keys," and "Modifier Tracks, Track References, and Alternate Tracks."

You should read this book if you are going to work with QuickTime movies.

Organization of This Document

This book consists of the following chapters:

- Movie Time and Space (page 11) describes the functions of the movie toolbox that your application will use to manipulate the timebase and spatial characteristics of movies, tracks, and media.
- Clock Components Overview (page 27) describes the general features and uses of clock components, and provides diagrams showing the relationship between clock components, applications, and the Movie Toolbox.
- Movie Clocks, Sound Clocks, and Video Output (page 29) describes how to set a movie's master clock, how a movie's default clock is chosen, the use of the sound clock, and how to use `SetMovieMasterClock` with video output components.
- Clock Component Functions (page 35) describes the functions which clock components must support, the callback functions that a clock component may optionally support, and functions which QuickTime uses to alert a clock component of changes in the environment. Applications programmers may be interested in the current time function and the callback functions. The other functions are of interest to developers who wish to create new components.
- Modifier Tracks (page 39) explains track references and modifier tracks, which create dynamic relationships between tracks in a QuickTime movie. You need to read this chapter if your application programmatically creates movies that contain modifier tracks, effects, filters, transitions, or alternate tracks.
- Movie Toolbox Access Keys (page 45) describes QuickTime's mechanism that lets an application that supplies data to register a password for the data with QuickTime and let a user enter the password to gain access to the data.
- Movie Posters and Movie Previews (page 47) describes the functions your application can use to work with posters and previews. A poster is a still frame from a movie, while a preview is a short excerpt.
- Previewing Files (page 49) describes how to create and display file previews shown as part of the Open File dialog. A file preview is typically a thumbnail image taken from the movie that is displayed in an Open File dialog. Some sample code is included. Read this if you would like to include preview thumbnails of movies.

See Also

The following Apple books cover related aspects of QuickTime programming:

- *QuickTime Overview* gives you the starting information you need to do QuickTime programming.
- *QuickTime Movie Basics* introduces you to some of the basic concepts you need to understand when working with QuickTime movies.
- *QuickTime Movie Creation Guide* describes some of the different ways your application can create a new QuickTime movie.
- *QuickTime Guide for Windows* provides information specific to programming for QuickTime on the Windows platform.
- *QuickTime API Reference* provides encyclopedic details of all the functions, callbacks, data types and structures, atom types, and constants in the QuickTime API.

Movie Time and Space

This chapter describes the functions of the movie toolbox that your application will use to manipulate the timebase and spatial characteristics of movies, tracks, and media.

Working With Movie Spatial Characteristics

The Movie Toolbox provides a number of functions that allow your application to determine and change the spatial characteristics of movies and tracks. Before using any of them, you should be familiar with the way in which the Movie Toolbox displays movies. Here are some tips for using these functions:

- You can use the `SetMovieGWorld` and `GetMovieGWorld` functions to work with a movie's graphics world.
- Your application can work with a movie's matrix by calling the `GetMovieMatrix` and `SetMovieMatrix` functions, and it can work with a track's matrix with the `GetTrackMatrix` and `SetTrackMatrix` functions. Then you can perform operations on matrices with the Movie Toolbox's matrix functions described in Matrix Functions (page 12).
- Certain functions affect the displayed movie and its tracks in the final display coordinate system. The `SetMovieGWorld` and `GetMovieGWorld` functions let you work with a movie's display destination. The `GetMovieBox` and `SetMovieBox` functions allow you to work with a movie's boundary rectangle and its associated transformations. Alternatively, you can use the `GetMovieMatrix` and `SetMovieMatrix` functions to work directly with a movie's transformation matrix. The `GetMovieDisplayBoundsRgn` function determines a movie's boundary region at the current movie time. On the other hand, the `GetMovieSegmentDisplayBoundsRgn` function determines a movie's boundary region over a specified time segment. You can use the `GetMovieDisplayClipRgn` and `SetMovieDisplayClipRgn` functions to work with a movie's display clipping region.
- The `GetTrackDisplayBoundsRgn` and `GetTrackSegmentDisplayBoundsRgn` functions determine a track's final boundary region. You can use the `GetTrackLayer` and `SetTrackLayer` functions to control the drawing order of tracks within a movie.
- A number of functions affect a movie's display boundaries before any display transformations. These functions operate in the movie's display coordinate system. You can use the `GetMovieClipRgn` and `SetMovieClipRgn` functions to work with a movie's clipping region; that is, the clipping region that is applied before the movie display transformation. Use the `GetMovieBoundsRgn` function to determine a movie's boundary region at the current movie time.
- Use the `GetTrackMovieBoundsRgn` function to work with a track's boundary region after matrix transformations have placed the track into the movie's display system. The `SetTrackMatrix` and `GetTrackMatrix` functions let you define a track's matrix transformations.
- The Movie Toolbox provides several functions that affect a track's display boundaries; these functions operate in the track's display coordinate system before any other display transformations are applied. The `GetTrackDimensions` and `SetTrackDimensions` functions allow you to establish a track's coordinate system and to establish a track's source rectangle.

- You can use the `GetTrackBoundsRgn` function to determine a track's boundary region. The `GetTrackClipRgn` and `SetTrackClipRgn` functions let you work with a track's clipping region. You can use the `GetTrackMatte` and `SetTrackMatte` functions to establish a track's matte. The `DisposeMatte` function allows you to dispose of a matte once you are finished with it.

Note: A track's source rectangle defines the coordinate system of the track. You specify the dimensions of the rectangle by providing the coordinates of the lower-right corner of the rectangle. The Movie Toolbox sets the upper-left corner to (0,0) in the track's coordinate system.

Matrix Functions

The Movie Toolbox provides a number of functions that allow you to work with transformation matrices. This section describes those functions. For descriptions of fixed-point and fixed-rectangle structures, see The Fixed-Point and Fixed-Rectangle Structures (page 12).

Note: The functions described in this section do not appear in the interface file `Movies.hMovie` Toolbox; rather, they appear in the `ImageCompression.h` interface file.

The Fixed-Point and Fixed-Rectangle Structures

The Movie Toolbox matrix functions provide two mechanisms for specifying points and rectangles. Some of the functions work with standard QuickDraw points and rectangles, which use integer values to identify coordinates. Others, such as the `TransformFixedRect` function, work with points and rectangles whose coordinates are expressed as fixed-point numbers. By using fixed-point numbers in these points and rectangles, the Movie Toolbox can support a greater degree of precision when defining graphic objects.

The `FixedPoint` data type defines a **fixed point.** The `FixedRect` data type defines a **fixed rectangle.** Note that both of these structures define the x coordinate before the y coordinate. This is different from the standard QuickDraw structures.

```
struct FixedPoint
{
    Fixed x;        /* point's x coordinate as fixed-point number */
    Fixed y;        /* point's y coordinate as fixed-point number */
};
typedef struct FixedPoint FixedPoint;
```

Field	Description
x	Defines the point's x coordinate as a fixed-point number.
y	Defines the point's y coordinate as a fixed-point number.

```
struct FixedRect
{
    Fixed left;         /* x coordinate of upper-left corner */
    Fixed top;          /* y coordinate of upper-left corner */
    Fixed right;        /* x coordinate of lower-right corner */
```

```
    Fixed bottom;       /* y coordinate of lower-right corner */
};
typedef struct FixedRect FixedRect;
```

Field	Description
`left`	Defines the x coordinate of the upper-left corner of the rectangle as a fixed-point number.
`top`	Defines the y coordinate of the upper-left corner of the rectangle as a fixed-point number.
`right`	Defines the x coordinate of the lower-right corner of the rectangle as a fixed-point number.
`bottom`	Defines the y coordinate of the lower-right corner of the rectangle as a fixed-point number.

Creating a Track Matte

Listing 1-1 provides an example of how to create a track matte. The `CreateTrackMatte` function adds an uninitialized, 8-bit-deep, grayscale matte to a track. The `UpdateTrackMatte` function draws a gray ramp rectangle around the edge of the matte and fills the center of the matte with black. (A ramp rectangle shades gradually from light to dark in smooth increments.)

Listing 1-1 Creating a track matte

```
void CreateTrackMatte (Track theTrack)
{
    QDErr err;
    GWorldPtr aGW;
    Rect trackBox;
    Fixed trackHeight;
    Fixed trackWidth;
    CTabHandle grayCTab;
    GetTrackDimensions (theTrack, &trackWidth, &trackHeight);
    SetRect (&trackBox, 0, 0, FixRound (trackWidth),
                FixRound (trackHeight));

    grayCTab = GetCTable(40);    /* 8 bit + 32 = 8 bit gray */
    err = NewGWorld (&aGW, 8, &trackBox, grayCTab,
                        (GDHandle) nil, 0);
    DisposeCTable (grayCTab);
    if (!err && (aGW != nil))
    {
        SetTrackMatte (theTrack, aGW->portPixMap);
        DisposeGWorld (aGW);
    }
}
void UpdateTrackMatte (Track theTrack)
{
    OSErr err;
    PixMapHandle trackMatte;
    PixMapHandle savePortPix;
    Movie       theMovie;
    GWorldPtr tempGW;
    CGrafPtr savePort;
```

```
    GDHandle saveGDevice;
    Rect        matteBox;
    short       i;

    theMovie = GetTrackMovie (theTrack);
    trackMatte = GetTrackMatte (theTrack);
    if (trackMatte == nil)
    {
        /* track doesn't have a matte, so give it one */
        CreateTrackMatte (theTrack);
        trackMatte = GetTrackMatte (theTrack);
        if (trackMatte == nil)
            return;
    }

    GetGWorld (&savePort, &saveGDevice);
    matteBox = (**trackMatte).bounds;
    err = NewGWorld(&tempGW,
                      (**trackMatte).pixelSize, &matteBox,
                      (**trackMatte).pmTable, (GDHandle) nil, 0);
    if (err || (tempGW == nil)) return;

    SetGWorld (tempGW, nil);
    savePortPix = tempGW->portPixMap;
    LockPixels (trackMatte);
    SetPortPix (trackMatte);

    /* draw a gray ramp rectangle around the edge of the matte */
    for (i = 0; i < 35; i++)
    {
        RGBColor aColor;
        long        tempLong;
        tempLong = 65536 - ((65536 / 35) * (long)i);
        aColor.red = aColor.green = aColor.blue = tempLong;
        RGBForeColor(&aColor);
        FrameRect (&matteBox);
        InsetRect (&matteBox, 1, 1);
    }

    /* fill the center of the matte with black */
    ForeColor (blackColor);
    PaintRect (&matteBox);

    SetPortPix (savePortPix);
    SetGWorld (savePort, saveGDevice);
    DisposeGWorld (tempGW);
    UnlockPixels (trackMatte);
    SetTrackMatte (theTrack, trackMatte);
    DisposeMatte (trackMatte);
}
```

Working With Movie Time

Every QuickTime movie has its own time base. A movie's time base allows all the tracks that make up the movie to be synchronized when the movie is played. The Movie Toolbox provides a number of functions that allow your application to determine and establish the time parameters of a movie:

- You can use the `GetMovieTimeBase` function to retrieve the time base for a movie.
- You can work with a movie's current time by calling the `GetMovieTime`, `SetMovieTime`, and `SetMovieTimeValue` functions.
- You can work with a movie's time scale by calling the `GetMovieTimeScale` and `SetMovieTimeScale` functions.
- The Movie Toolbox can calculate the total duration of a movie. You can use the `GetMovieDuration` function to retrieve a movie's duration.
- Your application can call the `GetMovieRate` and `SetMovieRate` to work with a movie's playback rate.

The next sections discuss those functions. Later sections in this chapter discuss the Movie Toolbox functions that allow you to work with the time parameters of tracks and media structures. For information about more functions that work with time, see Time Base Functions (page 18).

About Movie Time

At the most basic level, the Movie Toolbox allows you to process time based data. As such, the Movie Toolbox must provide a description of the time basis of that data as well as a definition of the context for evaluating that time basis. In QuickTime, a movie's time basis is referred to as its **time base.** Geometrically, you can think of the time base as a vector that defines the direction and velocity of time for a movie. The context for a time base is called its **time coordinate system.** Essentially, the time coordinate system defines the axis on which the time base vector is plotted. The smallest single unit of time marked on that axis is defined by the time scale as the units per absolute second.

Time Coordinate Systems

A movie's time coordinate system provides the context for evaluating the passage of time in the movie. If you think of the time coordinate system as defining an axis for measuring time, it is only natural that this axis would be marked with a scale that defines a basic unit of measurement. In QuickTime, that measurement system is called a **time scale.**

A QuickTime time scale defines the number of time units that pass each second in a given time coordinate system. A time coordinate system that has a time scale of 1 measures time in seconds. Similarly, a time coordinate system that has a time scale of 60 measures sixtieths of a second. In general, each time unit in a time coordinate system is equal to (1/time scale) seconds. Some common time scales are listed in Table 1-1.

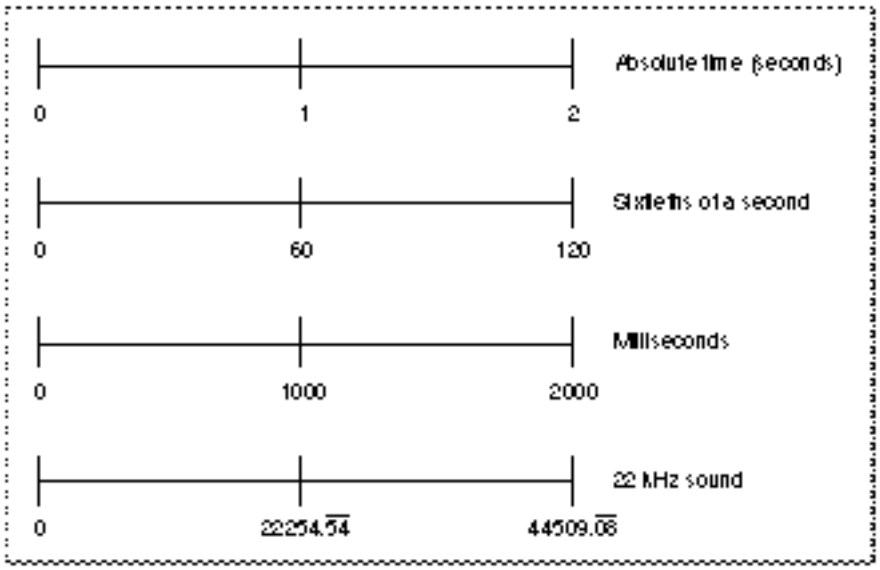
Table 1-1 Common movie time scales

Time scale	Absolute time measured
1	Seconds

Time scale	Absolute time measured
60	Sixtieths of a second (Macintosh ticks)
1000	Milliseconds
22254.54	Sound sampled at 22 kHz

Figure 1-1 shows a duration of two seconds in absolute time and equivalent durations in the common time scales listed in Table 1-1.

Figure 1-1 Time scales



A particular point in time in a time coordinate system is represented using a **time value.** A time value is expressed in terms of the time scale of its time coordinate system. Without an appropriate time scale, a time value is meaningless. For example, in a time coordinate system with a time scale of 60, a time value of 180 translates to 3 seconds. Because all time coordinate systems tie back to absolute time (that is, time as we measure it in seconds), the Movie Toolbox can translate time values from one time coordinate system into another.

Time coordinate systems have a finite maximum duration that defines the maximum time value for a time coordinate system (the minimum time value is always 0). Note that as a QuickTime movie is edited, the duration changes.

As the value of the time scale increases (as the time unit for a coordinate system gets smaller in terms of absolute time), the maximum absolute time that can be represented in a time coordinate system decreases. For example, if a time value were represented as an unsigned 16-bit integer, its maximum value would be 65,535. In a time coordinate system with a time scale of 1, the maximum time value would represent 65,535 seconds. However, in a time coordinate system with a time scale of 5, the maximum time value would correspond to 13,107 seconds. Hence, a time coordinate system's duration is limited by its time scale. QuickTime uses 32-bit and 64-bit quantities to represent time values, so you only need to worry about attaining a maximum absolute time in situations where a time coordinate system's duration is very long or its time scale is very large.

Time Bases

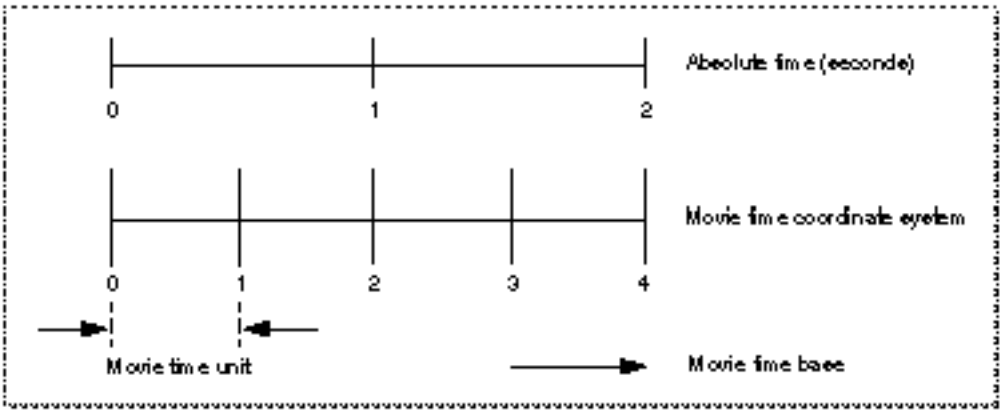
A movie's time base defines its current time value and the **rate** at which time passes for the movie. The rate specifies the speed and direction in which time travels in a movie. Negative rate values cause you to move backward through a movie's data; positive values move forward. The time base also contains a reference to the clock that provides timing for the time base. QuickTime clocks are implemented as components that are managed by the Component Manager.

Time bases exist independently of any specific time coordinate system. However, time values extracted from a time base are meaningless without a time scale. Therefore, whenever you obtain a time value from a time base, you must specify the time scale of the time value result. The Movie Toolbox translates the time base's time value into a value that is sensible in the specified time scale.

Note: A time base differs from a time coordinate system, which provides the foundation for a time base. (A time coordinate system is the field of play that defines the coordinate axis for a time base.) A time base operates in the context of a time coordinate system. It has a rate, which implies a direction as well as a speed through the movie.

Figure 1-2 represents a time coordinate system and a time base geometrically. The time coordinate system is represented by a coordinate axis. In this example, the time coordinate system has a time scale of 2; that is, there are two time units in each second. The duration of this time coordinate system is 2 seconds, which is equivalent to 4 time units. An object's time base is depicted by the large arrow under the axis that represents the time coordinate system. This time base has a current time value of 3 and a rate of 1. The starting time is a time value, expressed in the units of the time coordinate system.

Figure 1-2 A time coordinate system and a time base



Working With Track Time

The Movie Toolbox provides several functions that allow your application to determine and establish a track's time parameters. A track uses the time base of the movie that contains the track; therefore there are no functions that work with a track's time base or time scale. However, you can determine a track's duration and its offset from the start of a movie.

All of the tracks in a movie use the movie's time coordinate system. That is, the movie's time scale defines the basic time unit for each of the movie's tracks. Each track begins at the beginning of the movie, but the track's data might not begin until some time value other than 0. This intervening time is represented by blank space. In an audio track the blank space translates to silence; in a video track the blank space generates no visual image. This blank space is the **track offset.** Each track has its own **duration.** This duration need not correspond to the duration of the movie. A movie duration always equals the maximum track duration. These functions help you work with track time:

- You can use the `GetTrackDuration` function to determine a track's duration.
- The `SetTrackOffset` and `GetTrackOffset` functions enable you to work with a track's offset from the start of the movie that contains it.
- The `TrackTimeToMediaTime` function lets you translate a track's time to the corresponding time value of a media in the track.

Working With Media Time

The Movie Toolbox provides functions that allow your application to work with the time parameters of a media:

- You can use the `GetMediaDuration` function to determine a media's duration.
- The `GetMediaTimeScale` and `SetMediaTimeScale` let you determine or establish a media's time scale.

Time Base Functions

The Movie Toolbox provides a number of functions that allow you to work with time bases. A QuickTime time base defines the time coordinate system of a movie. However, you can also use QuickTime time bases to provide general timing services. This section describes the functions that allow your application to work with time bases.

This section has been divided into the following topics:

- Creating and Disposing of Time Bases (page 19) describes how to create and dispose of time bases and how to assign a time base to a movie
- Working With Time Base Values (page 19) discusses functions that allow your application to work with the contents of a time base
- Time Structures (page 20) describes a number of functions that allow you to convert times between time bases and to perform simple arithmetic on time values
- Time Base Callback Functions (page 21) describes the functions your application may use to condition a time base to invoke functions your application provides

Note: Time base functions do not change the value of the Movie Toolbox sticky error value.

Creating and Disposing of Time Bases

The following Movie Toolbox functions help your application create and dispose of time bases:

- The `NewTimeBase` function lets you create a new time base. You can use the `DisposeTimeBase` function to dispose of a time base once you are finished with it.
- Time bases rely on either a clock component or another time base for their time source. You can use the `SetTimeBaseMasterTimeBase` function to cause one time base to be based on another time base. The `GetTimeBaseMasterTimeBase` allows you to determine the master time base of a given time base.
- You can assign a clock component to a time base; that clock then acts as the master clock for the time base. You can use the `SetTimeBaseMasterClock` function to assign a clock component to a time base. The `GetTimeBaseMasterClock` function enables you to determine the clock component that is assigned to a time base. You can change the offset between a time base and its time source by calling the `SetTimeBaseZero` function.
- You can set the time source of a movie by calling the `SetMovieMasterTimeBase` and `SetMovieMasterClock` functions.

Note: Although most time base functions can be used at interrupt time, several of the Movie Toolbox functions cannot. For information about specific functions see the *QuickTime API Reference*.

Working With Time Base Values

Every time base contains a rate, a start time, a stop time, a current time, and some status information. The Movie Toolbox provides a number of functions that allow your application to work with the contents of a time base. You can use these functions:

- The `GetTimeBaseTime` function lets you retrieve the current time value of a time base. You can set the current time value by calling the `SetTimeBaseTime` function; this function requires you to provide a time structure. Alternatively, you can set the current time based on a time value by calling the `SetTimeBaseValue` function.
- You can determine the rate of a time base by calling the `GetTimeBaseRate` function. You can set the rate of a time base by calling the `SetTimeBaseRate` function. You can determine the effective rate of a specified time base (relative to the master time base to which it is subordinate) by calling the `GetTimeBaseEffectiveRate` function.
- You can retrieve the start time of a time base by calling the `GetTimeBaseStartTime` function. You can set the start time of a time base by calling the `SetTimeBaseStartTime` function. Similarly, you can use the `GetTimeBaseStopTime` and `SetTimeBaseStopTime` functions to work with the stop time of a time base.

The Movie Toolbox also provides functions that allow you to work with the status information of a time base:

- The `GetTimeBaseStatus` function allows you to read the current status of a time base.

- The `GetTimeBaseFlags` function helps you obtain the control flags of a time base. You can set these flags by calling the `SetTimeBaseFlags` function.

Time Structures

The Movie Toolbox provides a number of functions that allow you to work with time structures:

- You can use the `ConvertTime` function to convert a time you obtain from one time base into a time that is relative to another time base. Similarly, you can use the `ConvertTimeScale` function to convert a time from one time scale to another.
- You can add two times by calling the `AddTime` function; you can subtract two times with the `SubtractTime` function.

All of these functions work with time structures. You can use time structures to represent either time values or durations. Time values specify a point in time, relative to a given time base. Durations specify a span of time, relative to a given time scale. Durations are represented by time structures that have the time base set to 0 (that is, the `base` field in the time structure is set to `nil`).

The TimeRecord Structure

Many time management functions require that you place a time specification in a data structure called a `TimeRecord`.

```
struct TimeRecord
{
    CompTimeValue       value;      /* time value (duration or absolute) */
    TimeScale       scale;          /* units per second */
    TimeBase        base;           /* reference to the time base */
};
typedef struct TimeRecord TimeRecord;
```

Field	Description
`value`	Contains the time value. The time value defines either a duration or an absolute time by specifying the corresponding number of units of time. For durations, this is the number of time units in the period. For an absolute time, this is the number of time units since the beginning of the time coordinate system. The unit for this value is defined by the scale field. The time value is expressed as a `CompTimeValue` data type, which is a 64-bit integer quantity. This 64-bit quantity consists of two 32-bit integers, and it is defined by the `Int64` data type.
`scale`	Contains the time scale. This field specifies the number of units of time that pass each second. If you specify a value of 0, the time base uses its natural time scale.
`base`	Contains a reference to the time base. You obtain a time base by calling the Movie Toolbox's `GetMovieTimeBase` or `NewTimeBase` functions. If the time structure defines a duration, set this field to `nil`. Otherwise, this field must refer to a valid time base.

Large Time Values

You can specify the time value in a time structure as a 64-bit integer value as follows:

```
typedef Int64 CompTimeValue;
```

The Movie Toolbox uses this format so that extremely large time values can be represented. The `Int64` data type defines the format of these signed 64-bit integers.

```
struct Int64
{
    long hi;    /* high-order 32 bits-value field in time structure */
    long lo;    /* low-order 32 bits-value field in time structure */
};
typedef struct Int64 Int64;
```

Field	Description
`hi`	Contains the high-order 32 bits of the value. The high-order bit represents the sign of the 64-bit integer.
`lo`	Contains the low-order 32 bits of the value.

Time Base Callback Functions

If your application uses QuickTime time bases, it may define callback functions that are associated with a specific time base. Your application can then use these callback functions to perform activities that are triggered by temporal events, such as a certain time being reached or a specified rate being achieved. The time base functions of the Movie Toolbox interact with clock components to schedule the invocation of these callback functions; clock components are responsible for invoking the callback function at its scheduled time. Your application can use the functions described in this section to establish your own callback function and to schedule callback events.

You can define three types of callback events. These types are distinguished by the nature of the temporal event that triggers the Movie Toolbox to call your function. The three types are

- events that are triggered at a specified time
- events that are triggered when the rate reaches a specified value
- events that are triggered when the time value of a time base changes by an amount different from the time base's rate.

Use these functions to work with time base callbacks:

- You specify a callback event's type when you define the callback event, using the `NewCallBack` function.
- You specify whether your event can occur at interrupt time when you define the callback event, using the `NewCallBack` function. Your function is called closer to the triggering event at interrupt time, but it is subject to all the restrictions of interrupt functions (for example, your callback function cannot cause memory to be moved). If your function is not called at interrupt time, you are free of these restrictions; but your function may be called later, because the invocation is delayed to avoid interrupt time.

- The `NewCallBack` function allocates the memory to support a callback event. When you are done with the callback event, you dispose of it by calling the `DisposeCallBack` function.
- You schedule a callback event by calling the `CallMeWhen` function. Call `CancelCallBack` function to unschedule a callback event.
- You can retrieve the time base of a callback event by calling the `GetCallBackTimeBase` function.
- You can obtain the type of a callback event by calling the `GetCallBackType` function.

Time and Space Function Summary

Here is a summary of the Movie Toolbox functions you can use to manage movie time and spatial characteristics.

Managing Movie Space

The Movie Toolbox contains a number of functions that your application can use to determine or change the spatial display characteristics of movies and tracks. These functions affect the movie's graphics world, its matrix, the final display coordinate system, and the display boundaries. They include

- `SetTrackGWorld`
- `SetMovieGWorld`
- `GetMovieGWorld`
- `SetMovieBox`
- `GetMovieBox`
- `GetMovieDisplayBoundsRgn`
- `GetMovieSegmentDisplayBoundsRgn`
- `SetMovieDisplayClipRgn`
- `GetMovieDisplayClipRgn`
- `GetTrackSegmentDisplayBoundsRgn`
- `SetTrackLayer`
- `GetTrackLayer`
- `SetMovieMatrix`
- `GetMovieMatrix`
- `GetMovieBoundsRgn`
- `GetTrackMovieBoundsRgn`
- `GetMovieClipRgn`
- `SetTrackMatrix`
- `GetTrackMatrix`

- GetTrackBoundsRgn
- SetTrackDimensions
- GetTrackDimensions
- SetTrackClipRgn
- GetTrackClipRgn
- SetTrackMatte
- GetTrackMatte
- DisposeMatte
- SetMovieColorTable
- GetMovieColorTable

Working With Matrices

Several Movie Toolbox functions allow you to manipulate transformation matrices.

- SetIdentityMatrix
- GetMatrixType
- CopyMatrix
- EqualMatrix
- TranslateMatrix
- ScaleMatrix
- RotateMatrix
- SkewMatrix
- ConcatMatrix
- InverseMatrix
- TransformPoints
- TransformFixedPoints
- TransformRect
- TransformFixedRect
- TransformRgn
- RectMatrix
- MapMatrix

Managing Movie Time

Several functions are used to work with a movie's time parameters. Included are functions for retrieving the time base, working with the current movie time, working with the time scale, calculating the movie's duration, and getting and setting the playback rate.

- `GetMovieDuration`
- `SetMovieTimeValue`
- `SetMovieTime`
- `GetMovieTime`
- `SetMovieRate`
- `GetMovieRate`
- `SetMovieTimeScale`
- `GetMovieTimeScale`
- `GetMovieTimeBase`

Managing Track Time

Several functions work with a track's time parameters. All tracks share the movie's time base, but each track contains its own offset and duration. A function is also provided to translate track time into a value appropriate to the track's media.

- `GetTrackDuration`
- `SetTrackOffset`
- `GetTrackOffset`
- `TrackTimeToMediaTime`

Managing Media Time

Three functions work with a media's time parameters. Each media has its own time scale and duration.

- `GetMediaDuration`
- `SetMediaTimeScale`
- `GetMediaTimeScale`

Finding Interesting Times

You can call Movie Toolbox functions to search a movie, track or media for a particular sample, such as a keyframe. The functions return the time and duration of the next sample that meets the search criteria.

- `GetMovieNextInterestingTime`
- `GetTrackNextInterestingTime`
- `GetMediaNextInterestingTime`

Time Base Management

Various Movie Toolbox functions work with time bases. A QuickTime time base defines the time coordinate system for a movie. It can also be used to provide general timing services.

- Creating and Disposing of Time Bases (page 19)
  - `NewTimeBase`
  - `DisposeTimeBase`
  - `SetMovieMasterClock`
  - `SetMovieMasterTimeBase`
  - `SetTimeBaseMasterClock`
  - `GetTimeBaseMasterClock`
  - `SetTimeBaseMasterTimeBase`
  - `GetTimeBaseMasterTimeBase`
  - `SetTimeBaseZero`
- Working With Time Base Values (page 19)
  - `SetTimeBaseTime`
  - `SetTimeBaseValue`
  - `SetTimeBaseRate`
  - `GetTimeBaseRate`
  - `GetTimeBaseEffectiveRate`
  - `SetTimeBaseStartTime`
  - `GetTimeBaseStartTime`
  - `SetTimeBaseStopTime`
  - `GetTimeBaseStopTime`
  - `SetTimeBaseFlags`
  - `GetTimeBaseFlags`
  - `GetTimeBaseStatus`
- Time Structures (page 20)
  - `AddTime`
  - `SubtractTime`

- `ConvertTime`
- `ConvertTimeScale`

- Time Base Callback Functions (page 21)
  - `NewCallBack`
  - `CallMeWhen`
  - `CancelCallBack`
  - `DisposeCallBack`
  - `GetCallBackTimeBase`
  - `GetCallBackType`

Most of these functions are discussed in the preceding sections of this chapter.

Clock Components Overview

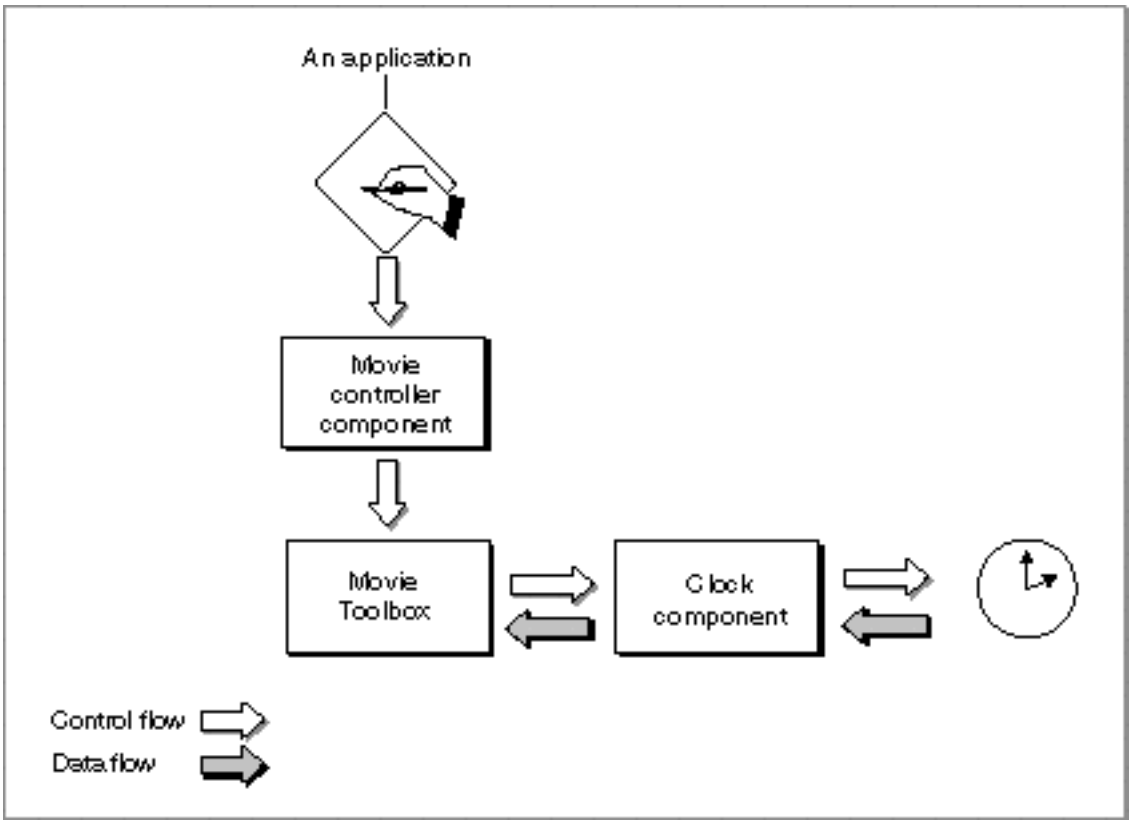
Clock components provide two basic services: they generate time information and schedule time-based callback events. QuickTime uses clock components to drive movie time bases. Applications can change the clock used for a movie's time base. Applications may also use clock components to schedule application-defined callback events.

Clock Components and Movie Time Bases

A clock component provides realtime clock information derived from an external source, typically a hardware source.

Figure 2-1 shows the relationships between an application, the movie controller component, the Movie Toolbox, and a clock component.

Figure 2-1 Clock component environment



Clock components are used by the Movie Toolbox for time bases. A time base defines a movie's current time value and the rate at which time passes for the movie. The rate specifies the speed and direction in which time travels in a movie. A time base also contains a reference to a clock which provides timing for the time base. The clock component used to drive a movie's time base is known as the **movie master clock.**

QuickTime chooses a default master clock, known as the **default clock,** depending on the movie media types. If you play a movie containing a sound track, QuickTime uses the sound clock by default. If the movie lacks a sound track, QuickTime uses the system clock as the default master clock.

You can obtain the clock component associated with a given media by calling `MediaGetClock`. You can also change the master clock associated with a movie's time base. For details, see Movie Clocks, Sound Clocks, and Video Output (page 29).

Interrupts and Callbacks

Your application can call any clock component function at interrupt time, except for the `ClockNewCallBack` and `ClockDisposeCallBack` functions. In addition, your application should not call the Component Manager's open or close component functions, such as `OpenComponent`, `OpenAComponent`, `OpenADefaultComponent`, or `CloseComponent`, at interrupt time.

Clock components may also support time-based callback events. The Movie Toolbox's time base functions allow applications and other programs to schedule functions to be called in specified circumstances. Since time bases derive their time information from clock components, ultimate responsibility for servicing these callback functions also falls to clock components.

The Movie Toolbox provides a set of support functions that clock component authors can use to manage callback events. These functions are described in Clock Component Functions (page 35).

A clock component is not required to support callback functions. Clock component authors can delegate this responsibility to another clock component. Component Capability Flags for Clocks (page 32) describes how a clock component can tell the Component Manager that the clock component does not support callback functions.

Movie Clocks, Sound Clocks, and Video Output

This chapter describes how to set a movie's master clock, how a movie's default clock is chosen, the use of the sound clock, how to use `SetMovieMasterClock` with video output components, and the component capability flags defined for clock components.

Overview

Clock components are used by the Movie Toolbox for time bases. A movie master clock is what drives a movie's time base.

A time base defines a movie's current time value and the rate at which time passes for the movie. The rate specifies the speed and direction in which time travels in a movie. A time base also contains a reference to a clock which provides timing for the time base.

When an application opens a movie containing a sound track, QuickTime uses the sound clock as the movie master clock. The sound clock is a clock component provided directly by the selected audio output device, or (if the audio output device doesn't provide one) a clock that simply watches the samples go by on their way out to the hardware and derives a clock from that.

In other words, if you play a movie containing audio and video media, the video will play at whatever rate the audio hardware is playing. If the movie lacks a sound track, QuickTime will use the system clock as the movie master clock.

The clock QuickTime chooses via this process is called the **default clock.**

You can obtain the clock component associated with a given media by calling `MediaGetClock`.

The component capability flags describe the capabilities of the clock component: specifically whether the clock keeps a constant rate (as opposed to drifting) and whether the component supports callback functions.

Setting and Changing the Master Clock

This section describes how to set and change the master clock associated with a movie time base.

You can obtain the clock component associated with a given media by calling `MediaGetClock`.

The ChooseMovieClock Function

`ChooseMovieClock` assigns a default clock to a movie. If you have changed a movie's master clock with `SetMovieMasterClock`, or would like to reset a movie's clock, use the `ChooseMovieClock` function.

Associated Components: The Video Output Component Clock

A Video Output Clock is the clock component associated with a specific video output component. This clock allows the time base used by a QuickTime movie to be driven by a specific output hardware device's clock, in order to synchronize video and sound when the output device is in use.

An application can ask for the clock component associated with the video output component and use this clock as a movie's master clock.

When using a Video Output Component, you can get an Instance of the Clock Component associated with the Video Output Component by calling `QTVideoOutputGetClock`.

Once you have this Clock Instance, it can be associated with a Movie by calling `SetMovieMasterClock`. Because a change to the display mode could affect a clock component, your application should call `QTVideoOutputGetClock` only between calls to the `QTVideoOutputBegin` and `QTVideoOutputEnd` functions.

When you want to reset the movie master clock back to the default clock, use `ChooseMovieClock`.

If you were previously using `SetMovieMasterClock` to reset a movies clock to the default clock, you should change to the new `ChooseMovieClock` method:

```
ChooseMovieClock(myMovie, 0);
```

Associated Components: Sound Output Component

A Sound Output Component is a software module that identifies, controls, and plays audio on a specific hardware device. Video output components, in addition to having a clock component, can have a Sound Output Component associated with them.

Developers can change the Sound Output Component used by a Media Handler by calling `MediaSetSoundOutputComponent`. This allows choosing between using the audio device associated with a video output device, another sound output device installed on the system, or the default sound output device.

Note: Calling `MediaSetSoundOutputComponent` can change the movie master clock.

Using the Sound Output Component Associated With the Video Output Component

To find the Sound Output Components associated with a Video Output Component use `QTVideoOutputGetIndSoundOutput`. Once the component is retrieved, call `MediaSetSoundOutputComponent` to set the sound output component for a media handler.

```
Component theSoundOut = 0;
ComponentInstance theVOutClock = NULL;
UnsignedFixed theSupportedAudioRate, myWantedAudioRate = eAudioRate48khz;
    ...

// Does this Video Output Component have a
```

```
// Sound Output Component associated with it?
if (ComponentFunctionImplemented(theInstance,
        kQTVideoOutputGetIndSoundOutputSelect)) {
    // Get the first sound output component associated
    // with the video output component
    err = QTVideoOutputGetIndSoundOutput(inVOComponentInstance,
                1, &theSoundOut);
    if (err || 0 == theSoundOut) goto bail;

    // Not all sound output components support all sample
    // rates, use GetSoundOutputInfo with the siSampleRateAvailable
    // selector and figure it out
    theSupportedAudioRate = MyChooseAudioRate(myWantedAudioRate,
                                              theSoundOut);

    // Set the sample rate for the audio output
    err = SoundComponentSetInfo((ComponentInstance)theSoundOut,
                    NULL, siSampleRate, (void *)theSupportedAudioRate);
    if (err) goto bail;

    // For each audio tracks media set the sound output component
    for (i = 0;i < theNumberAudioTracks; i++) {
        err = MediaSetSoundOutputComponent(inAudioMediaHandlers[i],
                                                    theSoundOut);
        if (err) goto bail;
    }
}

// Use the Video Output Clock as the Master Clock
// Set up the video output clock after sound or it
// gets set back to the default clock
if (ComponentFunctionImplemented(inVOComponentInstance,
                                 kQTVideoOutputGetClockSelect)) {
    err = QTVideoOutputGetClock(inVOComponentInstance, &theVOutClock);
    if (err || NULL == theVOutClock) goto bail;

    SetMovieMasterClock(inMovie, (Component)theVOutClock, NULL);
}
```

Switching Back to the Default Sound Output Component

To switch back to the Default Sound Output Component, use `MediaSetSoundOutputComponent` and pass in `NULL` for the Component parameter.

```
// Set the Sound Output back to the Default Sound Output
for (i = 0;i < theNumberAudioTracks; i++) {
    err = MediaSetSoundOutputComponent(theAudioMediaHandler[i], NULL);
    if (err) goto bail;
}

// Switch back to the default clock
ChooseMovieClock(myMovie, 0);
```

Choosing the Clock

As mentioned in the section Associated Components: Sound Output Component (page 30), choosing a Sound Output Component will reset the master clock. Therefore, once you choose the Sound Output Component, you should then set up the movie's master clock. You can either use the Video Output Clock (a logical choice when using a Video Output Component), or you could chose the default clock to provide audio and video sync for a movie.

When using the Video Output Clock, be sure to set the movie's clock back to the default clock before calling `QTVideoOutputEnd`.

If you want to use the Video Output Clock, call `SetMovieMasterClock` and pass in the Video Output Clock Instance.

```
// Use the Video Output Clock as the Master Clock
// Set up the video output clock after sound or it
// gets set back to the default clock
if (ComponentFunctionImplemented(theVOComponentInstance,
                                 kQTVideoOutputGetClockSelect)) {
    err = QTVideoOutputGetClock(gVOComponentInstance, &theVOutClock);
    if (err || NULL == theVOutClock) goto bail;

    SetMovieMasterClock(myMovie, (Component)theVOutClock, NULL);
}
```

When choosing or switching to the Default Clock, use `ChooseMovieClock`.

```
// Use the default clock
ChooseMovieClock(myMovie, 0);
```

Remember, not all sound devices have clocks, and not all video output components have clock components associated with them, so be sure to check.

`SetMovieMasterClock` and `ChooseMovieClock` will cancel each other out; the last API called is the one that sets the clock.

`MediaSetSoundOutputComponent` can change the movie master clock.

Component Capability Flags for Clocks

The Component Manager allows a clock component to specify information about its capabilities in the `componentFlags` field of the component description structure. Apple has defined two component flags for clock components. These flags specify information about the capabilities of the clock component. The component author sets these flags in the `componentFlags` field of the component's `ComponentDescription` structure. The following constants can be used to manipulate these flags. Clock component authors should set them appropriately for their clock.

```
enum {
    kClockRateIsLinear = 1,        /* clock keeps constant rate */
    kClockImplementsCallBacks = 2  /* clock supports callback events */
};
```

`kClockRateIsLinear` indicates that your clock maintains a constant rate. Most clocks that you deal with in the everyday world fall into this category. An example of a clock with an irregular rate is a clock that is dependent on the position of the Macintosh computer's mouse; the clock's rate might change depending upon where the user moves the mouse. Set this flag to 1 if your clock has a constant rate.

`kClockImplementsCallBacks` indicates that your clock supports callback events. Set this flag to 1 if your clock supports callback events.

You should set the `componentFlags` field appropriately in the component description structure that is associated with your clock component.

Clock Component Functions

This section describes the functions that are provided by clock components. These functions are described from the perspective of the Movie Toolbox, the entity that is most likely to call clock components. If you are developing a clock component, your component must behave as described here.

This section has been divided into the following topics:

- Getting the Current Time (page 36) describes the function that allows QuickTime to obtain the current time from a clock component. This function can also be used by applications.
- Using the Callback Functions (page 36) discusses the functions that allow clock components to help applications define and schedule time base callback functions.
- Managing the Time (page 36) describes functions that help clock components manage their time correctly.

If you are developing an application that uses clock components, you should read the next section, Getting the Current Time (page 36).

If you are developing a clock component, you need to be familiar with all the functions described in this section.

You can use the following constants to refer to the request codes for each of the functions that your clock component must support:

Note: Your application can call any clock component function at interrupt time, except for the `ClockNewCallBack` and `ClockDisposeCallBack` functions. In addition, your application should not call the Component Manager's `OpenComponent` and `CloseComponent` functions at interrupt time.

```
/* constants to refer to request codes for supported functions */
enum {
    kClockGetTimeSelect                     = 0x1,  /* ClockGetTime */
    kClockNewCallBackSelect                 = 0x2,/* ClockNewCallBack */
    kClockDisposeCallBackSelect             = 0x3,/* ClockDisposeCallBack */
    kClockCallMeWhenSelect                  = 0x4,/* ClockCallMeWhen */
    kClockCancelCallBackSelect              = 0x5,/* ClockCancelCallBack */
    kClockRateChangedSelect                 = 0x6,/* ClockRateChanged */
    kClockTimeChangedSelect                 = 0x7,/* ClockTimeChanged */
    kClockSetTimeBaseSelect                 = 0x8,/* ClockSetTimeBase */
    kClockStartStopChangedSelect            = 0x9,/* ClockStartStopChanged */
    kClockGetRateSelect                     = 0xA /* ClockGetRate */
};
```

Getting the Current Time

Clock components provide a single function, `ClockGetTime`, that allows the Movie Toolbox to obtain the current time.

Using the Callback Functions

Applications that use QuickTime time bases may define callback functions that are associated with a specific time base. Applications can then use these callback functions to perform activities that are triggered by temporal events, such as a certain time being reached or a specified rate being achieved. The time base functions of the Movie Toolbox interact with clock components to schedule the invocation of these callback functions; your clock component is responsible for calling the callback function at its scheduled time.

The functions described in this section are called by the Movie Toolbox to support applications that define time base callback functions. Note that your clock component can delegate its callback events to another component by calling the Component Manager's `DelegateComponent` function.

The `ClockNewCallBack` function allows your clock component to allocate the memory to support a new callback event. When an application discards a callback event, the Movie Toolbox calls your clock component's `ClockDisposeCallBack` function.

The Movie Toolbox calls your clock component's `ClockCallMeWhen` function when an application wants to schedule a callback event. When the callback function is to be invoked to service the event, the Movie Toolbox calls your component's `ClockCancelCallBack` function so that you can remove the callback event from the list of scheduled events.

Managing the Time

Clock components provide several functions that allow the Movie Toolbox to alert your component to changes in its environment. Three of these functions, `ClockTimeChanged`, `ClockRateChanged`, and `ClockStartStopChanged`, are associated with application callback functions and help your component determine whether to invoke the callback function. The fourth, the `ClockSetTimeBase` function, tells your clock component about the time base it is supporting.

Movie Toolbox Clock Support Functions

The Movie Toolbox provides a number of support functions for clock components. All of these functions help your component manage its associated callback functions. Your clock component may call any of these functions at interrupt time. These functions should only be called by clock components:

- Use the `AddCallBackToTimeBase` function to add a callback event to the list of scheduled callback events maintained by the Movie Toolbox. You should use the `RemoveCallBackFromTimeBase` function to remove a callback event from the list.

- When your clock component determines that it is time to invoke a callback function, you should use the `ExecuteCallBack` function to cause the Movie Toolbox to call the function.
- If your clock component needs to scan all its associated callback events, you can use the `GetFirstCallBack` and `GetNextCallBack` functions.

Data Types

The clock component data structure is a private data structure. Programs that use your clock component never change the contents of this data structure directly. Your clock component provides functions that allow programs to use this data structure.

The callback header structure specifies the callback function for an operation. Your application can obtain callback function identifiers by calling its clock component's `ClockNewCallBack` function.

The `QTCallBackHeader` data type defines the callback header structure.

```
struct QTCallBackHeader {
    long    callBackFlags;    /* flags used by clock component to communicate
                                 scheduling data about callback to QuickTime */

    long    reserved1;        /* reserved for use by Apple */
    char    qtPrivate[40];    /* reserved for use by Apple */
};
```

Field	Description
`callBackFlags`	Contains flags that your component can use to communicate scheduling information about the callback event to the Movie Toolbox. This scheduling information tells the Movie Toolbox what time base events your clock component needs to know about in order to support the callback event.
`reserved1`	Reserved for use by Apple.
`qtPrivate`	Reserved for use by Apple.

The following flags are defined for the `callBackFlags` field (all other flags must be set to 0):

```
enum {
    qtcbNeedsRateChanges = 1,       /* clock needs to know about rate
    qtcbNeedsTimeChanges = 2        /* clock needs to know about time
    qtcbNeedsStartStopChanges = 4   /* clock needs to know about
                                       time base changes */
};
```

Flag	Description
`qtcbNeedsRateChanges`	Indicates that your clock component needs to know about rate changes. If you set this flag to 1, QuickTime calls your ClockRateChanged function whenever the rate of the callback event's time base changes.

Flag	Description
`qtcbNeedsTimeChanges`	Indicates that your clock component needs to know about time changes. If you set this flag to 1, the Movie Toolbox calls your ClockTimeChanged function whenever a program changes the time value of the time base, or when the time value changes by an amount that is different from the time base's rate.
`qtcbNeedsStart-StopChanges`	Indicates that your clock component needs to know about the time base's start and stop changes. If you set this flag to 1, the Movie Toolbox calls your `ClockStartStopChanged` function whenever a program changes the start or stop time of the time base.

Component Types for Clocks

Apple has defined a type value and a number of subtype values for clock components. All clock components have a component type value of `'clok'`. The component subtype value indicates the type of clock. You can use the following constants to specify these type and subtype values.

```
#define clockComponentType        'clok'     /* clock component type */
#define systemTickClock           'tick'     /* system tick clock */
#define systemSecondClock         'seco'     /* system seconds clock */
#define systemMillisecondClock    'mill'     /* system millisecond clock */
#define systemMicrosecondClock    'micr'     /* system microsecond clock */
```

Modifier Tracks

Track references and modifier tracks create dynamic relationships between tracks in a QuickTime movie. Track references describe the relationship between tracks; for example, one track's output can be connected to an input of another track, or two tracks may be related as alternate versions, to be used in different languages. Track references are used to create modifier tracks, alternate tracks, and chapter lists, and to relate timecode tracks to segments of other tracks.

A modifier track does not present its data as a video or audio track normally does; instead it uses its data to modify the presentation of other tracks. For example, the output of a video track can used as the source of image data for a sprite track or effect track, or the output of a tween track can be used as the source of the volume setting of an audio track. This chapter discusses several special modifier track types, including:

- **Alternate tracks.** These are typically different versions of a track to be used in different circumstances, such as multiple text or sound tracks in different languages, or different bit-rate video tracks for optimal display on faster or slower computers.
- **Chapter lists,** sets of named entry points into a movie that can be jumped to either programmatically or through the user interface.
- **Timecode tracks.** They contain timecode information, typically gathered from a source external to QuickTime (such as SMPTE timecodes). They relate this timecode data to segments of other QuickTime tracks, such as video, audio, and text tracks.

This chapter assumes that you are familiar with QuickTime tracks in general and the QuickTime functions used to determine and change the display characteristics of movies and tracks.

Track References

Track references are a feature of QuickTime that allows you to relate a movie's tracks to one another. The QuickTime track-reference mechanismsupports many-to-many relationships. That is, any movie track may contain one or more track references, and any track may be related to one or more other tracks in the movie.

Track references can be useful in a variety of ways. For example, track references can be used to relate timecode tracks to other movie tracks. Another use of track references is to associate one or more text tracks that contain subtitles with the appropriate audio track or tracks.

Track references are also used to create alternate tracks, as discussed in Working With Alternate Tracks (page 43).

Every movie track contains a list of its track references. Each track reference identifies another, related track. That related track is identified by its track identifier. The track reference itself contains information that allows you to classify the references by type. This type information is stored in an `OSType` data type. You are free to specify any type value you want. Note, however, that Apple has reserved all lowercase type values.

You may create as many track references as you want, and you may create more than one reference of a given type. Each track reference of a given type is assigned an index value. The index values start at 1 for each different reference type. QuickTime maintains these index values so that they always start at 1 and count by 1.

Functions That Manipulate Track References

The following functions help you work with track references:

- The `AddTrackReference` function allows you to relate one track to another.
- The `DeleteTrackReference` function removes that relationship.
- The `SetTrackReference` and `GetTrackReference` functions allow you to modify an existing track reference so that it identifies a different track.
- The `GetNextTrackReferenceType` and `GetTrackReferenceCount` functions allow you to scan all of a track's track references.

Creating Modifier Tracks

Modifier tracks are used to modify other tracks. Redirecting the output of a modifier track to the input of another track lets you create a wide variety of effects, such as panning audio or presenting video data as a sprite.

Any QuickTime track can be made into a modifier track. For example, you can use the output of a video track as a sprite's image source, or the output of two video tracks as inputs to a transition effect in an effect track.

To create a modifier track, you need to use track references and input maps. A track reference lets you specify which track receives a modifier track's output. An input map specifies the part or parts of the receiving track to be modified by the output of the modifier track. For example, the output of a tween track might modify a sound track's volume, balance, or both.

Kinds of Modifier Tracks

A modifier track does not present any data; instead, it sends data to another track that uses that information to modify how it presents its own data. Any track can be either a sender or a presenter, but not both. For example, instead of simply playing video, a video track can send its image data to a sprite track. The sprite track then uses that video data to replace the image of one of its sprites. When the movie is played, the video track appears as a sprite.

Another use of modifier tracks is to store a series of sound volume levels, which is what occurs when you work with a tween track. These sound levels can be sent to a sound track as it plays, to dynamically adjust the volume. A similar use of modifier tracks is to store location and size information. This data can be sent to a video track to cause it to move and resize as it plays.

Because a modifier track can send its data to more than one track, you can easily synchronize actions between multiple tracks. For example, a single modifier track containing matrices as its samples can make two separate video tracks follow the same path.

Limitations of Spatial Modifier Tracks

A modifier track may cause a track to move outside of its original boundary regions. This may present problems, since applications do not expect the dimensions or location of a QuickTime movie to change over time.

To ensure that a movie maintains a constant location and size, QuickTime limits the area in which a spatially modified track can be displayed. A movie's "natural" shape is defined by the region returned by `GetMovieBoundsRgn`. QuickTime clips all spatially modified tracks against the region returned by `GetMovieBoundsRgn`. This means that a track can move outside of its initial boundary regions, but it cannot move beyond the combined initial boundary regions of all tracks in the movie. Areas uncovered by a moving track are handled in the same way as areas uncovered by tracks with empty edits.

If a track has to move through a larger area than that defined by the movie's boundary region, the movie's boundary region can be enlarged to any desired size by creating a spatial track (such as a video track) of the desired size but with no data. As long as the track is enabled, it contributes to the boundary regions of the movie.

Media Handler Support

The video, base, and tween media handlers can send the data they receive to other tracks. The text media handler can also send data, but none of the track media handlers can currently receive it. The sound, music, and 3D media handlers do not support sending their data to other tracks.

Not all media handlers support all input types. Media handlers can decide which input types to support. Table 5-1 lists the input types supported by each Apple-supplied media handler.

Table 5-1 Input types supported by Apple-supplied media handlers

Input type	Video	Text	Sound	MPEG	Music	Sprite	Timecode	3D
Matrix	X	X		X		X	X	X
Graphics mode	X	X		X		X	X	X
Clip	X	X		X		X	X	X
Volume			X	X	X			
Balance			X	X	X			
Sprite image						X		X
3D sound			X		X			

Creating Movies With Modifier Tracks

To create a movie with modifier tracks, first you create a movie with all the desired tracks, then you create the modifier track. To link the modifier track to the track that it modifies, use the `AddTrackReference` function shown in Listing 5-1.

Listing 5-1 Linking a modifier track to the track it modifies

```
long addedIndex;
AddTrackReference   (aVideoTrack,
                     aModifierTrack,
                     kTrackModifierReference,
                     &addedIndex);
```

The reference doesn't completely describe the modifier track's relationship to the track it modifies. Instead, the reference simply tells the modifier track to send its data to the specified track. The receiving track doesn't know what it should do with that data. A single track may also be receiving data from more than one modifier track. You have to update the media input map of the receiving track to specify how the data is to be interpreted.

Manipulating Media Input Maps

Each track has particular attributes such as size, position, and volume associated with it. The media input map of that track describes where the variable parameters are stored so that modifier tracks know where to send their data. When a track is copied, its input map is also copied. `CopyTrackSettings` also transfers the media input map.

QuickTime provides two functions you can use to maintain media input maps:

- `GetMediaInputMap`
- `SetMediaInputMap`

To describe how each modifier input should be used, each track's media has an input map. The media's input map describes how the data being sent to each input of a track should be interpreted by the receiving track. After creating the reference, it is necessary to update the receiving track's media input map. When `AddTrackReference` is called, it returns the index of the reference added. That index is the index of the input that needs to be described in the media input map. If the modifier track created above contains regions to change the shape of the video track, the code shown in Listing 5-2 updates the input map appropriately.

Listing 5-2 Updating the input map

```
QTAtomContainer inputMap;
QTAtom inputAtom;
OSType inputType;
Media aVideoMedia = GetTrackMedia(aVideoTrack);
GetMediaInputMap (aVideoMedia, &inputMap);
QTInsertChild(
          inputMap, kParentAtomIsContainer,
          kTrackModifierInput,
          addedIndex,
          0,
          0,
          nil,
          &inputAtom);
inputType = kTrackModifierTypeClip;
QTInsertChild (
          inputMap,
          inputAtom,
          kTrackModifierType,
```

```
            1,
            0,
            sizeof(inputType),
            &inputType,
            nil);
SetMediaInputMap(aVideoMedia, inputMap);
QTDisposeAtomContainer(inputMap);
```

The media input map allows you to store additional information for each input. In the preceding example, only the type of the input is specified. In other types of references, you may need to specify additional data.

When a modifier track is sending an empty track edit, or is disabled or deleted, all receiving tracks are notified that the track input is inactive. When an input becomes inactive, it is reset to its default value. For example, if a track is receiving data from a clip modifier track and that input becomes inactive, the shape of the track reverts to the shape it would have if there were no clip modifier track.

Working With Alternate Tracks

QuickTime lets you define alternate tracks in a movie. You can use alternate tracks to support multiple languages or to present different levels of visual quality in the movie. You collect alternate tracks into groups. Alternate track groups are collections of tracks that represent some conceptual data but are appropriate for use in different play environments. For example, you might have some 4-bit data in one track and some 8-bit data in another. Working with alternate tracks allows you to set up alternatives from which QuickTime can choose.

QuickTime selects one track from each alternate group when it plays the movie. For example, you could create a movie that has three separate audio tracks: one in English, one in French, and one in Spanish. You would collect these audio tracks into an alternate group. When the user plays the movie, QuickTime selects the track from this group that corresponds to the current language setting for the movie.

Similarly, you can use alternate tracks to store data of different quality. When the user plays the movie, QuickTime selects the track that best suits the capabilities of the user's computer on which the movie is being played. In this manner, you can create a single movie that can accommodate the playback characteristics of a number of different computer configurations.

QuickTime allows you to store quality information for media structures that are assigned to either sound or video tracks. For all tracks, QuickTime uses bits 6 and 7 of the quality setting. These bits encode a relative quality value. These values range from 0 to 3. You can use higher quality values to indicate larger sample sizes. For example, consider a movie that has two sound tracks that are alternates for each other: one contains 8-bit sound and the other contains 16-bit sound. You could assign a quality value of `mediaQualityNormal` to the 8-bit media and a value of `mediaQualityBetter` to the 16-bit media. QuickTime would play the 16-bit media only if the user's configuration could handle 16-bit sound. Otherwise, QuickTime would use the 8-bit media. The sound media handler determines the sample size for each sound media for QuickTime by examining the media's sound description structure.

In addition, QuickTime also uses bits 0 through 5 (the low-order bits) of the quality setting. You use these bits to indicate the pixel depths at which the media should be played. Each bit corresponds to a single depth value, ranging from 1-bit pixels to 32-bit pixels. You may use these bits to control the playback of both video and sound tracks.

As an example, consider a movie that contains three video tracks with the following characteristics:

Track	Characterics
A	1-bit video data, no compression
B	Compressed using the Apple Video Compressor
C	Compressed using the Joint Photographic Experts Group (JPEG) compressor

You could assign the following quality values to these track's media structures:

Track	Quality value
A	`MediaQualityDraft` + 1-bit depth + 2-bit depth (quality value is 0x0003: 0x0000 + 0x0003)
B	`MediaQualityNormal` + 4-bit depth + 8-bit depth + 16-bit depth + 32-bit depth (quality value is 0x007C: 0x0040 + 0x003C)
C	`MediaQualityBetter` + 4-bit depth + 8-bit depth + 16-bit depth + 32-bit depth (quality value is 0x00BC: 0x0080 + 0x003C)

QuickTime would always use Track A when playing the movie on 1-bit and 2-bit displays. At the other pixel depths, the video media handler determines which track to use by examining the availability and performance of the specified decompressors. If the JPEG decompressor can play back at full frame rate, QuickTime would use Track C. Otherwise, QuickTime uses Track B. The video media handler determines the compressor that is appropriate for each media by examining the media's image description structure.

You set a movie's language by calling the `SetMovieLanguage` function.

To establish alternate groups of tracks, you can use the `SetTrackAlternate` and `GetTrackAlternate` functions.

You can work with the language and quality characteristics of media by calling the `GetMediaLanguage`, `SetMediaLanguage`, `GetMediaQuality`, and `SetMediaQuality` functions.

By default, QuickTime automatically selects the appropriate tracks to play according to a movie's quality and language settings, as well as the capabilities of the user's computer. Whenever your application calls theSetMovieGWorld, `SetMovieBox`, `UpdateMovie`, or `SetMovieMatrix` function, QuickTime checks each alternate group for an appropriate track. However, you can control this selection process. Use the `SetAutoTrackAlternatesEnabled` function to enable or disable automatic track selection. The `SelectMovieAlternates` function instructs QuickTime to select appropriate tracks immediately. If no tracks in an alternate track group are enabled, then QuickTime does not activate any track from that group during automatic track selection.

Movie Toolbox Access Keys

Access keys make it possible to protect data. They allow an application that supplies data to register a password for the data with QuickTime and allows a user to enter the password to gain access to the data. For example, a codec can protect data it compresses with a password, so that it is available only to someone with the password. Similarly, the creator of a movie can require a password to view the movie.

In order to gain access to protected data, the user enters the access key in the QuickTime Settings control panel.

System and Application Access Keys

There are two kinds of access keys:

- **System access keys;** data protected by a system access key can be unlocked by any QuickTime caller on the computer.
- **Application access keys;** data protected by an application access key can be unlocked only by QuickTime clients for that application.

System access keys are useful for data that needs to be used by more than one application. When a system access key is registered for data, applications do not have to perform any additional registration to unlock the data. In contrast, application access keys are normally registered by the application in which the data is available, and each application registers the application access keys it uses. For example, a CD-ROM vendor can register the same application access key for all the data on the CD-ROM, which makes the data available to the application on the CD-ROM (such as a game) and inaccessible to all other applications. This prevents browsing of the data by users.

Access Key Types

Access keys are grouped by type. For example, there could be an access key type defined specifically for the Cinepak codec. Grouping access keys lets a QuickTime caller request only those keys that apply to it. This speeds operations involving large numbers of keys which might otherwise interfere with the performance of real-time operations. The functions for using access keys all require an access type.

Using Access Keys

This section illustrates how to use access keys.

Registering an Access Key

Listing 6-1 illustrates how to register an application access key.

Listing 6-1 Registering an application access key

```
OSErr myErr = 0;
Str255 keyType = doomCDKeyType;
long flags = AccessKeySystemFlag;
handle keyHdl;
keyHdl = NewHandle (sizeof("keykeykey")-1);
/* put key in handle */
myErr = QTRegisterAccessKey (keyType, flags, keyHdl);
```

Listing 6-2 illustrates how to register a system access key.

Listing 6-2 Registering a system access key

```
OSErr myErr = 0;
Str255 keyType = doomCDKeyType;
long flags = 0;
handle keyHdl;
keyHdl = NewHandle (sizeof("keykeykey")-1);
/* put key in handle */
myErr = QTRegisterAccessKey (keyType, flags, keyHdl);
```

Getting Access Keys

Listing 6-3 illustrates how to get application access keys of a particular type.

Listing 6-3 Getting access keys

```
OSErr myErr = 0;
Str255 keyType = doomCDKeyType;
long flags = 0;
handle keyHdl;
/* handle initialization here */
myErr = QTGetAccessKeys (keyType, flags, keyHdl);
```

Unregistering an Access Key

Listing 6-4 illustrates how to unregister a system access key.

Listing 6-4 Unregistering an access key

```
OSErr myErr = 0;
Str255 keyType = doomCDKeyType;
long flags = AccessKeySystemFlag;
handle keyHdl;
keyHdl = NewHandle (sizeof("keykeykey")-1);
/* put key in handle */
myErr = QTUnRegisterAccessKey (keyType, flags, keyHdl);
```

Movie Posters and Movie Previews

A QuickTime movie may contain a preview and a poster:

- A movie preview is a very short version of a movie, typically less than five seconds in duration. It is intended to give the user an idea of a movie's contents.
- A movie poster is a still frame representing the movie.

The following Movie Toolbox functions allow your application to work with movie previews and movie posters:

- Use the `PlayMoviePreview` function to display a movie's preview. The `PlayMoviePreview` function sets the movie into preview mode, plays the movie preview, sets the movie back to normal playback mode, and returns to your application.
- Alternatively, your application can control the playback of a movie's preview. Use the `SetMoviePreviewMode` function to place a movie into preview mode. You can then use the `StartMovie` and `StopMovie` functions, described below, to control movie playback. Your application can find out if a movie is in preview mode by calling the `GetMoviePreviewMode` function.
- Your application can specify the starting time and duration of the movie preview with the `SetMoviePreviewTime` and `GetMoviePreviewTime` functions.
- Use the `ShowMoviePoster` function to display a movie's poster. You can work with the poster's boundary rectangle using the `SetPosterBox` and `GetPosterBox` functions. Your application can work with the starting time of the poster with the `SetMoviePosterTime` and `GetMoviePosterTime` functions. Posters always have no duration.
- Tracks may be specified for use in the movie, its preview, its poster, or any combination of the three. So, for example, when the Movie Toolbox plays the movie preview it uses only those tracks that are assigned to the preview. Your application controls the use of a movie's tracks with the `SetTrackUsage` function. You can find out how a track is used by calling the `GetTrackUsage` function.

Controlling Movie Playback of Previews

This section describes a number of high-level functions provided by the Movie Toolbox that allow your application to play movies. For information about how to control a movie's playback rate, see Controlling Movie Time (page 48).

You can use the `StartMovie` and `StopMovie` functions to start and stop movies.

The Movie Toolbox provides functions that can be used to control your position within a movie. You can use two functions, `GoToBeginningOfMovie` and `GoToEndOfMovie`, to set the position at either the beginning or the end of a movie. These functions are described in this section. Functions that work with time bases, such as `SetMovieTimeValue` and `GetMovieTimeScale`, can be used to control the current position anywhere within a movie.

Controlling Movie Time

Every QuickTime movie has its own time base. A movie's time base allows all the tracks that make up the movie to be synchronized when the movie is played. The Movie Toolbox provides a number of functions that allow your application to determine and establish the time parameters of a movie:

- You can use the `GetMovieTimeBase` function to retrieve the time base for a movie.
- You can work with a movie's current time by calling the `GetMovieTime`, `SetMovieTime`, and `SetMovieTimeValue` functions.
- You can work with a movie's time scale by calling the `GetMovieTimeScale` and `SetMovieTimeScale` functions.
- The Movie Toolbox can calculate the total duration of a movie. You can use the `GetMovieDuration` function to retrieve a movie's duration.
- Your application can call the `GetMovieRate` and `SetMovieRate` to work with a movie's playback rate.

Previewing Files

QuickTime includes extensions to the Standard File Package that allow you to create and display file previews, information that gives the user an idea of a file's contents without opening the file. Typically, a file's preview is a small PICT image (called a *thumbnail*), but previews may also contain other types of information that is appropriate to the type of file being considered. For example, a text file's preview might tell the user when the file was created and what it discusses. You can use the Image Compression Manager to create thumbnail images.

Standard File Functions

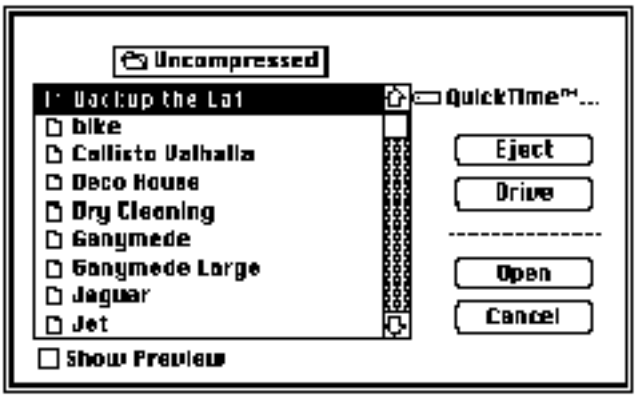
QuickTime provides standard file functions that your application can use to display a file's preview during the Open dialog box. These functions allow your application to support previews automatically.

Note: Before using these new standard file functions, make sure that the Image Compression Manager is installed.

The Movie Toolbox provides two standard file functions that allow you to display file previews in an Open dialog box using standard file reply structures: `SFGetFilePreview` and `SFPGetFilePreview`. The `SFGetFilePreview` function corresponds to the existing `SFGetFile` function; the `SFPGetFilePreview` function corresponds to the existing `SFPGetFile` function.

The `SFGetFilePreview` function uses the dialog box shown in Figure 8-1. The `SFPGetFilePreview` function can also use this dialog box, if you do not supply your own.

Figure 8-1 SFGetFilePreview or SFPGetFilePreview dialog box without preview



You use these new functions in place of the existing standard file functions to indicate whether or not you want to allow the user to display previews during the Open dialog box. The user displays a file's preview by selecting a file in the dialog box and clicking Show Preview. When the user does so, the functions display the preview for the file, as shown in Figure 8-2.

Figure 8-2 SFGetFilePreview or SFPGetFilePreview dialog box with preview

The preview area of the dialog box is displayed whenever previewing is enabled.

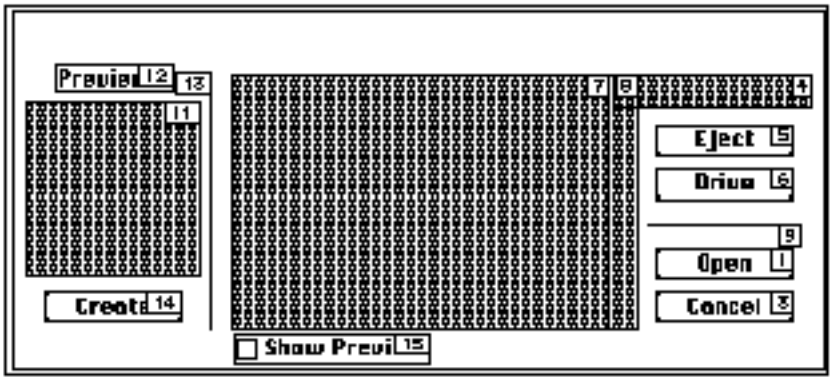
Customizing Your Interface

If your application requires it, you can customize the user interface for identifying files. The `SFGetFilePreview` function does not allow you to use a custom dialog box by creating your own dialog template resource. However, the `SFPGetFilePreview` function does let you access a custom dialog box of any resource type with the `dlgID` parameter.

Figure 8-3 shows the standard dialog box used by `SFPGetFilePreview` and `SFGetFilePreview`. Your dialog box and dialog filter function must support at least these dialog items.

Note: Alter the dialog boxes only if necessary. Apple does not guarantee future compatibility if you use a customized dialog box.

Figure 8-3 Standard preview dialog box for SFGetFilePreview and SFPGetFilePreview



Items to the left of item 13 are visible only when previewing. If you want to define items that are visible only during a file preview, place them to the left of item 13 in your custom dialog box.

If your application defines a custom dialog box, be sure to include the following items in your dialog box definition:

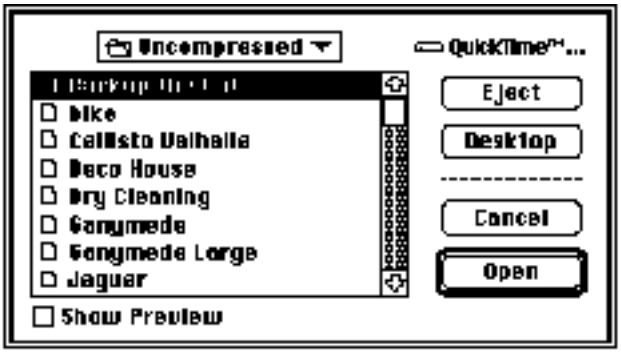
```
enum
{
/* dialog items to include in dialog box definition for use
   with SFPGetFilePreview function
*/
    sfpItemPreviewAreaUser              = 11,       /* user preview area */
    sfpItemPreviewStaticText            = 12,       /* static text preview */
    sfpItemPreviewDividerUser           = 13,       /* user divider preview */
    sfpItemCreatePreviewButton          = 14,       /* create preview button */
    sfpItemShowPreviewButton            = 15        /* show preview button */
};
```

Previewing Files Using Standard File Reply Structures

The Movie Toolbox provides two standard file functions, `standardGetFilePreview` and `CustomGetFilePreview`, that allow you to display file previews in an Open dialog box using standard file reply structures (of type `StandardFileReply`). The `StandardGetFilePreview` function corresponds to the existing `StandardGetFile` function; the `CustomGetFilePreview` function corresponds to the existing `CustomGetFile` function. Both of these new functions take the same parameters as their existing counterparts.

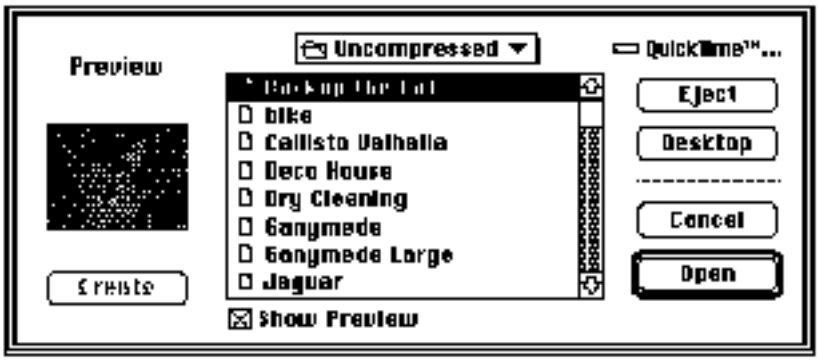
The `StandardGetFilePreview` function uses the dialog box shown in Figure 8-4. The `CustomGetFilePreview` function can also use this dialog box, if you do not supply your own.

Figure 8-4 StandardGetFilePreview or CustomGetFilePreview dialog box without preview



You use these new functions in place of the existing standard file functions whenever you want to allow the user to display previews during the Open dialog box. The user causes a file's preview to be displayed by selecting a file in the dialog box and clicking Show Preview. When the user does so, the functions display the preview for the file, as shown in Figure 8-5.

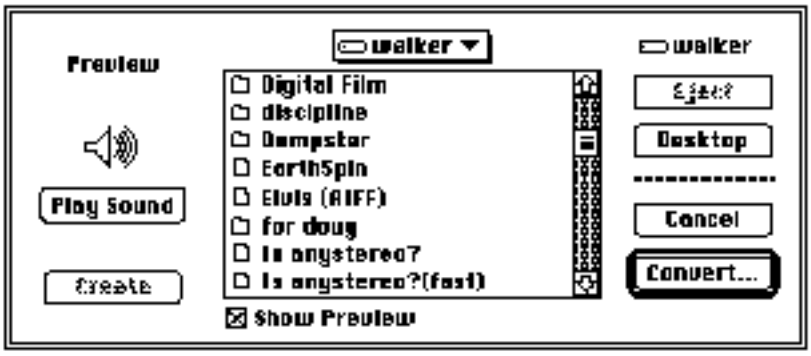
Figure 8-5 StandardGetFilePreview or CustomGetFilePreview dialog box with preview



The preview portion of the dialog box is displayed only when the dialog box is showing a file's preview.

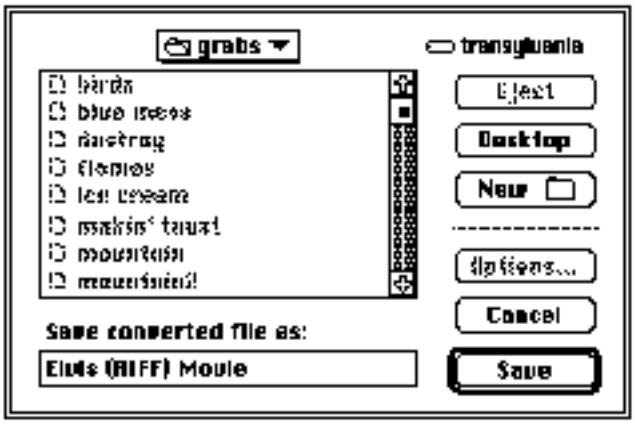
The `SFGetFilePreview`, `SFPGetFilePreview`, `StandardGetFilePreview`, and `CustomGetFilePreview` functions allow the user to automatically convert files to movies if your application requests movies. If there is a file that can be converted into a movie file using a movie import component, then the file is shown in the Standard File dialog box in addition to any movies. When the user selects the file, the Open button changes to a Convert button. Figure 8-6 provides an example of this dialog box.

Figure 8-6 Dialog box showing automatic file-to-movie conversion option



Choosing Convert displays a dialog box that allows the user to choose where the converted file should be saved. Figure 8-7 shows this dialog box.

Figure 8-7 Dialog box for saving a movie converted from a file



When conversion is complete, the converted file is returned to the calling application as the movie that the user chose. If you want to disable automatic file conversion in your application, you must write a file filter function and pass it to the file preview display function you are using. Your file filter function must call the File Manager's `FSpGetFileInfo` function on each file that is passed to it to determine its actual file type. If the File System parameter block pointer passed to your file filter function indicates that the file type is `'MooV'`, and the actual type returned by `FSpGetFileInfo` is not `'MooV'`, then the file filter function will convert this file. If you do not wish a file to be displayed as a candidate for conversion, your file filter function should return a value of `true` when it is called for that file.

A file filter function filters the files that are displayed to the user in a dialog box. You specify this function in the `fileFilter` parameter of the `SFGetFilePreview`, `StandardGetFilePreview`, and `CustomGetFilePreview` routines. If this parameter is not `nil`, `SFGetFilePreview` calls the function for each file to determine whether to display the file to the user. The `SFGetFilePreview` function supplies you with the information it receives from the File Manager's `GetFileInfo` routine.

Functions for Creating File Previews

The Movie Toolbox provides two functions that allow you to create file previews. File previews contain information that gives the user an idea of a file's contents without opening the file. Typically, a file's preview is a small PICT image (called a *thumbnail*), but previews may also contain other types of information that is appropriate to the type of file being considered. For example, a text file's preview might tell the user when the file was created and what it discusses.

Note: The `MakeFilePreview` and `AddFilePreview` functions documented in this section are not listed in the `Movies`.h interface file; rather, they appear in the `ImageCompression`.h interface file.

You can use the `MakeFilePreview` function to create a preview for a file. The `AddFilePreview` function allows you to add a preview that you have created to a file.

Generating Pictures From Movies

The Movie Toolbox provides a set of functions that allow your application to create QuickDraw pictures from movies, tracks, and posters. This section discusses those functions.

You can use the `GetMoviePict` function to create a picture from a movie or its preview; you can use the `GetTrackPict` function to create a picture from a track. The `GetMoviePosterPict` function lets you create a picture that contains a movie's poster. If a movie or track has no spatial representation, the returned picture is empty (that is, the upper-left and lower-right coordinates are equal).

When memory is low, the `GetMoviePict` function now reports out of memory errors instead of returning empty pictures.

Document Revision History

This table describes the changes to *QuickTime Movie Internals Guide*.

Date	Notes
2006-01-10	New document that describes the technology inside QuickTime movies, including time management, modifier tracks, access keys, posters, and movie and file previews.
	Replaces "Movie Toolbox: Time and Spatial Characteristics," "Clock Components," "Movie Toolbox: Access Keys," "Movie Toolbox: Previews," and "Modifier Tracks, Track References, and Alternate Tracks."
2002-09-17	New document that explains how to manipulate the temporal and spatial characteristics of movies, tracks, and media.

Document Revision History