

Movie Toolbox Reference

This section describes all the Movie Toolbox data types and functions. The Movie Toolbox provides a rich and varied set of functions that allow your application to work with QuickTime movies. This discussion has been divided into the following sections:

- “Data Types” identifies the data types used by your application when interacting with the Movie Toolbox
- “Functions for Getting and Playing Movies” describes the functions that applications can use to create, get, and play movies
- “Functions That Modify Movie Properties” describes functions that allow you to change the display, time, and sound characteristics of a movie
- “Functions for Editing Movies” discusses the functions that you can use to edit the contents of movies
- “Media Functions” discusses the functions that allow you to communicate with media handlers
- “Functions for Creating File Previews” describes the functions provided by the Movie Toolbox that allow you to create file previews
- “Functions for Displaying File Previews” describes the Movie Toolbox functions that let you display file previews
- “Time Base Functions” discusses the various Movie Toolbox functions that work with time bases
- “Matrix Functions” describes the Movie Toolbox functions that allow you to manipulate transformation matrices
- “Application-Defined Functions” describes the functions your application can provide when interacting with the Movie Toolbox

If you are developing a QuickTime-aware application that plays existing movies, you should read “Functions for Getting and Playing Movies,” which begins on page 2-81.

If you are developing an application that allows the user to create and edit movies, you should also read “Functions for Editing Movies,” which begins on page 2-242. More advanced display and editing applications may use some of the functions described in “Functions That Modify Movie Properties,” which begins on page 2-157.

Data Types

Most Movie Toolbox data structures are private data structures. Your application never modifies the contents of these structures directly. Rather, the Movie Toolbox provides a number of functions that allow you to work with these data structures.

Movie Identifiers

You identify a data structure to the Movie Toolbox by means of a data type that is supplied by the Movie Toolbox. The following data types are currently defined:

Media	Specifies the media for an operation. Your application obtains a media identifier from such Movie Toolbox functions as <code>NewTrackMedia</code> and <code>GetTrackMedia</code> (described on page 2-153 and page 2-206, respectively).
Movie	Specifies the movie for an operation. Your application obtains a movie identifier from such functions as <code>NewMovie</code> , <code>NewMovieFromFile</code> , and <code>NewMovieFromHandle</code> (described on page 2-92, page 2-88, and page 2-90, respectively).
MovieEditState	Specifies the movie edit state for an operation. Your application obtains a movie edit state identifier when you create the edit state by calling the <code>NewMovieEditState</code> function (described on page 2-255).
QTCallback	Specifies the callback for an operation. You obtain a callback identifier from the <code>NewCallback</code> function (described on page 2-336).
TimeBase	Specifies the time base for an operation. Your application obtains a time base identifier from the <code>NewTimeBase</code> or <code>GetMovieTimeBase</code> functions (described on page 2-316 and page 2-190, respectively).
Track	Specifies the track for an operation. Your application obtains a track identifier from such Movie Toolbox functions as <code>NewMovieTrack</code> and <code>GetMovieTrack</code> (described on page 2-151 and page 2-204, respectively).
TrackEditState	Specifies the track edit state for an operation. Your application obtains a track edit state identifier when you create the edit state by calling the <code>NewTrackEditState</code> function (described on page 2-269).
UserData	Specifies the user data list for an operation. You obtain a user data list identifier by calling the <code>GetMovieUserData</code> , <code>GetTrackUserData</code> , or <code>GetMediaUserData</code> functions (described on page 2-231, page 2-232, and page 2-233, respectively).

The Time Structure

The Movie Toolbox provides a number of functions that allow you to work with time specifications. These functions are described in “Time Base Functions” beginning on page 2-315. Many of these functions require that you place a time specification in a data structure called a *time structure*. The time structure allows you to fully describe a time specification. The `TimeRecord` data type defines the format of a time structure.

```
struct TimeRecord
{
    CompTimeValue  value;    /* time value (duration or absolute) */
    TimeScale      scale;    /* units per second */
}
```

Movie Toolbox

```

    TimeBase    base;          /* reference to the time base */
};
typedef struct TimeRecord TimeRecord;

```

Field descriptions

value	<p>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.</p> <p>The time value is expressed as a <code>CompTimeValue</code> 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 <code>Int64</code> data type, which is described next in this section.</p>
scale	<p>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.</p>
base	<p>Contains a reference to the time base. You obtain a time base by calling the Movie Toolbox's <code>GetMovieTimeBase</code> or <code>NewTimeBase</code> functions (described on page 2-190 and page 2-316, respectively).</p> <p>If the time structure defines a duration, set this field to <code>nil</code>. Otherwise, this field must refer to a valid time base.</p>

You specify the time value in a time structure in 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 descriptions

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.

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 (described on page 2-349), 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 descriptions

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 descriptions

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.

The Sound Description Structure

A sound description structure contains information that defines the characteristics of one or more sound samples. Data in the sound description structure indicates the type of compression that was used, the sample size, the rate at which samples were obtained, and so on. Sound media handlers use the information in the sound description structure when they process the sound samples.

Movie Toolbox

See the chapter “Image Compression Manager” for a description of the image description structure, which contains information that defines the characteristics of an image.

The `SoundDescription` data type defines the layout of a sound description structure. See “Media Functions,” which begins on page 2-281, for more information about sound media handlers.

```
struct SoundDescription
{
    long  descSize;      /* number of bytes in this structure */
    long  dataFormat;    /* format of the sound data */
    long  resvd1;        /* reserved--set to 0 */
    short resvd2;        /* reserved--set to 0 */
    short dataRefIndex;  /* reserved--set to 1 */
    short version;       /* reserved--set to 0 */
    short revlevel;      /* reserved--set to 0 */
    long  vendor;        /* reserved--set to 0 */
    short numChannels;   /* number of channels used by sample */
    short sampleSize;    /* number of bits in each sample */
    short compressionID; /* reserved--set to 0 */
    short packetSize;    /* reserved--set to 0 */
    Fixed sampleRate;    /* rate at which samples were obtained */
};
```

Field descriptions

<code>descSize</code>	Defines the total size, in bytes, of this sound description structure.
<code>dataFormat</code>	Describes the format of the sound data. Possible values include: <ul style="list-style-type: none"> 'raw ' Sound samples are stored uncompressed, in offset-binary format (that is, sample data values range from 0 to 255). 'twos' Sound samples are stored uncompressed, in twos-complement format (that is, sample data values range from -128 to 127). The Sound Manager uses this format when it creates sound files in Audio Interchange File Format (AIFF). 'MAC3 ' Sound samples have been compressed by the Sound Manager at a ratio of 3:1. 'MAC6 ' Sound samples have been compressed by the Sound Manager at a ratio of 6:1. <p>Some older movie files sometimes have a zero value in this field. You should assume that this is the same as the 'raw ' value.</p>
<code>resvd1</code>	Reserved for Apple. Set this field to 0 in any sound description structures you create.
<code>resvd2</code>	Reserved for Apple. Set this field to 0 in any sound description structures you create.

Movie Toolbox

<code>dataRefIndex</code>	Reserved for Apple. Set this field to 0 in any sound description structures you create.										
<code>version</code>	Reserved for Apple. Set this field to 0 in any sound description structures you create.										
<code>revLevel</code>	Reserved for Apple. Set this field to 0 in any sound description structures you create.										
<code>vendor</code>	Reserved for Apple. Set this field to 0 in any sound description structures you create.										
<code>numChannels</code>	Indicates the number of sound channels used by the sound sample. Set this field to 1 for monaural sounds; set it to 2 for stereo sounds.										
<code>sampleSize</code>	Specifies the number of bits in each sound sample. Set this field to 8 for 8-bit sound; set it to 16 for 16-bit sound.										
<code>compressionID</code>	Reserved for Apple. Set this field to 0 in any sound description structures you create.										
<code>packetSize</code>	Reserved for Apple. Set this field to 0 in any sound description structures you create.										
<code>sampleRate</code>	Indicates the rate at which the sound samples were obtained. Sound media handlers use this value to influence the natural playback speed of the sound described by this sound description structure. This field contains an unsigned, fixed-point number that specifies the number of samples collected per second. Some common values include: <table> <tr> <td><code>0x15BBA2E8</code></td><td>Specifies a sample rate of 5563.6363 samples per second.</td></tr> <tr> <td><code>0x1CFA2E8B</code></td><td>Specifies a sample rate of 7418.1818 samples per second.</td></tr> <tr> <td><code>0x2B7745D1</code></td><td>Specifies a sample rate of 11127.2727 samples per second.</td></tr> <tr> <td><code>0x56EE8BA3</code></td><td>Specifies a sample rate of 22254.5454 samples per second.</td></tr> <tr> <td><code>0xAC440000</code></td><td>Specifies a sample rate of 44100.0000 samples per second.</td></tr> </table>	<code>0x15BBA2E8</code>	Specifies a sample rate of 5563.6363 samples per second.	<code>0x1CFA2E8B</code>	Specifies a sample rate of 7418.1818 samples per second.	<code>0x2B7745D1</code>	Specifies a sample rate of 11127.2727 samples per second.	<code>0x56EE8BA3</code>	Specifies a sample rate of 22254.5454 samples per second.	<code>0xAC440000</code>	Specifies a sample rate of 44100.0000 samples per second.
<code>0x15BBA2E8</code>	Specifies a sample rate of 5563.6363 samples per second.										
<code>0x1CFA2E8B</code>	Specifies a sample rate of 7418.1818 samples per second.										
<code>0x2B7745D1</code>	Specifies a sample rate of 11127.2727 samples per second.										
<code>0x56EE8BA3</code>	Specifies a sample rate of 22254.5454 samples per second.										
<code>0xAC440000</code>	Specifies a sample rate of 44100.0000 samples per second.										

Functions for Getting and Playing Movies

The Movie Toolbox provides a number of functions that allow applications to get and play movies. There are also a number of functions that allow you to create new movies. This section describes those functions and has been divided into the following topics:

- “Initializing the Movie Toolbox” discusses the functions that your application must use to gain access to the Movie Toolbox
- “Error Functions” discusses the Movie Toolbox functions that allow you to work with error codes returned by Movie Toolbox functions
- “Movie Functions” describes functions that your application can use to create and access movie resources and movie files

Movie Toolbox

- “Saving Movies” describes the Movie Toolbox functions that allow you to save movies
- “Controlling Movie Playback” describes the functions that you can use to control movie playback
- “Movie Posters and Movie Previews” discusses the functions that allow applications to work with movie posters and movie previews
- “Movies and Your Event Loop” discusses the Movie Toolbox functions that your application must call from its main event loop
- “Preferred Movie Settings” describes functions your application can use to set the preferred playback settings of a movie
- “Enhancing Movie Playback Performance” discusses several techniques for improving movie playback performance
- “Disabling Movies and Tracks” describes the functions that allow your application to disable movies and tracks
- “Generating Pictures From Movies” discusses the Movie Toolbox functions that allow your application to create pictures from movie data
- “Creating Tracks and Media Structures” describes the functions your application must use to create new data for a movie
- “Working With Progress and Cover Functions” describes the functions that allow you to specify a custom function that is called during movie playback

Initializing the Movie Toolbox

The Movie Toolbox maintains state information for every application that is currently using the toolbox. The toolbox uses this information to keep track of the application’s movies. Before calling any other Movie Toolbox functions, your application must establish this working environment by calling the `EnterMovies` function. When your application is finished with the Movie Toolbox, you can release this storage by calling the `ExitMovies` function.

EnterMovies

Before you call any Movie Toolbox functions, you must initialize the toolbox. Use the `EnterMovies` function to initialize the Movie Toolbox. When your application calls this function, the Movie Toolbox creates its private storage area for your application.

You should initialize any other Macintosh managers your application uses before calling the `EnterMovies` function.

```
pascal OSErr EnterMovies (void);
```

DESCRIPTION

If the `EnterMovies` function fails, it returns an error value—be sure to check the value returned by this function before using any other facilities of the Movie Toolbox.

In addition, you should use the Gestalt Manager to determine whether the Movie Toolbox is installed (see “Determining Whether the Movie Toolbox Is Installed” beginning on page 2-33 for more information).

Your application may call the `EnterMovies` function multiple times for a given A5 world, as long as you balance each invocation of `EnterMovies` with an invocation of `ExitMovies`.

SPECIAL CONSIDERATIONS

The Movie Toolbox identifies an application by the value in the A5 register. If you are writing a stand-alone code resource, you must ensure that A5 is the same whenever you call any Movie Toolbox functions.

ERROR CODES

Memory Manager errors

SEE ALSO

Listing 2-3 on page 2-39 provides an example of the `EnterMovies` function.

ExitMovies

QuickTime calls the `ExitMovies` function automatically when your application quits—you only need to call this function if you finish with the Movie Toolbox long before your application is ready to quit. As a general rule, your application should not use this function.

```
pascal void ExitMovies (void);
```

DESCRIPTION

When you call the `ExitMovies` function, the Movie Toolbox releases the private storage (which may be significant) that was allocated when you called the `EnterMovies` function, which is described in the previous section.

SPECIAL CONSIDERATIONS

Before calling the `ExitMovies` function, be sure that you have closed your connections to any components that use the Movie Toolbox (such as movie controllers, sequence grabbers, and so on).

ERROR CODES

None

Error Functions

The Movie Toolbox provides a number of functions that allow your application to examine result codes generated by toolbox functions. In addition, the Movie Toolbox allows your application to provide a function that performs custom error notification. This section discusses these error functions.

IMPORTANT

The Movie Toolbox introduces an additional error-reporting mechanism. In addition to returning errors as function results, the Movie Toolbox functions return error indications to calling applications by setting one of two values that are private to the Movie Toolbox: a current error value or a sticky error value. Your application can retrieve these values by calling the `GetMoviesError` or `GetMoviesStickyError` functions described in this section. To let you know whether there is an error indication, the heading “ERROR CODES” may appear with the entry “None” in function descriptions throughout this chapter. ▲

The Movie Toolbox maintains two error values for your application: the current error and the sticky error. The **current error** value contains the result code from the last Movie Toolbox function. The toolbox updates the current error value each time your application calls a Movie Toolbox function. Your application may call the `GetMoviesError` function to obtain the current error value after calling any Movie Toolbox function. Many Movie Toolbox functions do not return an error as a function result—you must use the `GetMoviesError` function to obtain the result code. Even if a function explicitly returns an error as a function result, that result is also available using the `GetMoviesError` function.

The Movie Toolbox saves a result code in the **sticky error** value. Your application clears the sticky error value by calling the `ClearMoviesStickyError` function. The Movie Toolbox then places the first nonzero result code from any toolbox function used by your application into the sticky error value. The Movie Toolbox does not replace the value in the sticky error value until your application clears the value again. Your application uses the `GetMoviesStickyError` function to obtain the result code stored in the sticky error value. In this manner, you can preserve and retrieve important result code information.

Your application uses the `SetMoviesErrorProc` function to designate an error function. The Movie Toolbox calls this error function each time there is an error.

GetMoviesError

The `GetMoviesError` function returns the contents of the current error value and resets the current error value to 0.

```
pascal OSErr GetMoviesError (void);
```

DESCRIPTION

The current error value contains the result code from the previous Movie Toolbox function. Most Movie Toolbox functions do not return an error as a function result—you must use the `GetMoviesError` function to obtain the result code. Even if a function explicitly returns an error as a function result, that result is also available using the `GetMoviesError` function.

ERROR CODES

Any Movie Toolbox result code (see “Summary of the Movie Toolbox” at the end of this chapter)

GetMoviesStickyError

The `GetMoviesStickyError` function returns the contents of the sticky error value. The sticky error value contains the first nonzero result code from any Movie Toolbox function that you called after having cleared the sticky error with the `ClearMoviesStickyError` function.

```
pascal OSErr GetMoviesStickyError (void);
```

DESCRIPTION

The Movie Toolbox does not clear the sticky error value when you call the `GetMoviesStickyError` function. Your application clears the sticky error value by calling the `ClearMoviesStickyError` function, which is described in the next section.

ERROR CODES

Any Movie Toolbox result code (see “Summary of the Movie Toolbox” at the end of this chapter)

ClearMoviesStickyError

The `ClearMoviesStickyError` function clears the sticky error value.

```
pascal void ClearMoviesStickyError (void);
```

DESCRIPTION

The Movie Toolbox does not place a result code into the sticky error value until the field has been cleared. Your application should clear the sticky error value to ensure that it does not contain a stale result code.

ERROR CODES

None

SetMoviesErrorProc

The Movie Toolbox allows applications to perform custom error notification. Your application must identify its custom error-notification function to the Movie Toolbox. The `SetMoviesErrorProc` function allows you to identify your application's error-notification function. Error-notification functions can be especially useful when you are debugging your program.

```
pascal void SetMoviesErrorProc (ErrorProcPtr errProc,
                                long refcon);
```

`errProcPtr`

Points to your error-notification function, `MyErrProc`.

The entry point to your error-notification function must take the following form:

```
pascal void MyErrProc (OSErr theErr, long refCon);
```

See “Application-Defined Functions” beginning on page 2-354 for details on the parameters.

`refcon`

Contains a reference constant value. The Movie Toolbox passes this reference constant to your error-notification function each time it calls your function.

DESCRIPTION

Once you have identified an error-notification function, the Movie Toolbox calls your function each time the current error value is to be set to a nonzero value. The Movie Toolbox manages the sticky error value. The Movie Toolbox calls your error-notification function only in response to errors generated by the Movie Toolbox.

SPECIAL CONSIDERATIONS

The `SetMoviesErrorProc` function is just for debugging.

ERROR CODES

None

Movie Functions

The Movie Toolbox provides a set of functions that allow your application to create, access, and convert movie files. Movie files contain data for QuickTime movies. You can also use the Movie Toolbox to load movies into memory, in preparation for working with the movie. These functions differ based on where the movie is stored.

Before your application can play a movie, you must first open the file that contains the movie. Your application can use the `OpenMovieFile` function (described on page 2-98) to open a movie file. Once you are done with the file, your application releases the file by calling the `CloseMovieFile` function. Your application can create and open a new movie file by calling the `CreateMovieFile` function. Your application can delete a movie file by calling the `DeleteMovieFile` function.

You can use the `NewMovie` function to create a new empty movie. If your application is loading a movie from an existing file, use either the `NewMovieFromFile` function or the `NewMovieFromDataFork` function. The `NewMovieFromFile` function works with the file reference number you obtain from the `OpenMovieFile` function. The `NewMovieFromDataFork` function works with movies stored in your document file's data fork. Your application can then use the functions described in "Saving Movies," which begins on page 2-100, to load and store movies.

You can use the `ConvertFileToMovieFile` function to specify an input file and convert it to a movie file. The `ConvertMovieToFile` takes a specified movie (or a single track within that movie) and converts it into an output file.

Once you are finished working with a movie, you should release the resources used by the movie by calling the `DisposeMovie` function.

NewMovieFromFile

The `NewMovieFromFile` function creates a movie in memory from a resource that is stored in a movie file. Your application specifies the movie file with the file reference number that was returned by the `OpenMovieFile` function, which is described on page 2-98. Your application can use the `NewMovieFromHandle` function, described in the next section, to load a movie from a handle. Once you have opened a movie file and loaded a movie, your application can proceed to work with the movie.

```
pascal OSErr NewMovieFromFile (Movie *theMovie, short resRefNum,
                                short *resId,
                                StringPtr resName,
                                short newMovieFlags,
                                Boolean *dataRefWasChanged);
```

theMovie Contains a pointer to a field that is to receive the new movie's identifier. If the function cannot load the movie, the returned identifier is set to `nil`.

resRefNum Identifies the movie file from which the movie is to be loaded. Your application obtains this value from the `OpenMovieFile` function, described on page 2-98.

resId Contains a pointer to a field that specifies the resource containing the movie data that is to be loaded. If the field referred to by the `resId` parameter is set to 0, the Movie Toolbox loads the first movie resource it finds in the specified file. The toolbox then returns the movie's resource ID number in the field referred to by the `resId` parameter. The following enumerated constant is available:

`movieInDataForkResID`

Forces the movie to come out of the data fork. If the resource was stored in the file's data fork, the Movie Toolbox sets the returned value to `movieInDataForkResID` (-1). In this case, you cannot add a movie resource to the file unless you create a resource fork in the movie file.

If the `resId` parameter is set to `nil`, the Movie Toolbox loads the first movie resource it finds in the specified file and does not return that resource's ID number.

resName Points to a character string that is to receive the name of the movie resource that is loaded. If you set the `resName` parameter to `nil`, the toolbox does not return the resource name.

Movie Toolbox

`newMovieFlags`

Controls the operation of the `NewMovieFromFile` function. The following flags are available (be sure to set unused flags to 0):

`newMovieActive`

Controls whether the new movie is active. Set this flag to 1 to make the new movie active. You can make a movie active or inactive by calling the `SetMovieActive` function, which is described on page 2-145.

`newMovieDontResolveDataRefs`

Controls how completely the Movie Toolbox resolves data references in the movie resource. If you set this flag to 0, the toolbox tries to completely resolve all data references in the resource. This may involve searching for files on multiple volumes. If you set this flag to 1, the Movie Toolbox only looks in the specified file.

If the Movie Toolbox cannot completely resolve all the data references, it still returns a valid movie identifier. In this case, the Movie Toolbox also sets the current error value to `couldNotResolveDataRef`.

`newMovieDontAskUnresolvedDataRefs`

Controls whether the Movie Toolbox asks the user to locate files. If you set this flag to 0, the Movie Toolbox asks the user to locate files that it cannot find. If the Movie Toolbox cannot locate a file even with the user's help, the function returns a valid movie identifier and sets the current error value to `couldNotResolveDataRef`.

`newMovieDontAutoAlternate`

Controls whether the Movie Toolbox automatically selects enabled tracks from alternate track groups. If you set this flag to 1, the Movie Toolbox does not automatically select tracks for the movie—you must enable tracks yourself.

`dataRefWasChanged`

Contains a pointer to a Boolean value. The Movie Toolbox sets the Boolean to indicate whether it had to change any data references while resolving them. The toolbox sets the Boolean value to `true` if any references were changed. Use the `UpdateMovieResource` function (described on page 2-103) to preserve these changes.

Set the `dataRefWasChanged` parameter to `nil` if you do not want to receive this information. See “Creating Tracks and Media Structures” beginning on page 2-150 for more information about data references.

Movie Toolbox

DESCRIPTION

The Movie Toolbox sets many movie characteristics to default values. If you want to change these defaults, your application must call other Movie Toolbox functions. For example, the Movie Toolbox sets the movie's graphics world to the one that is active when you call `NewMovieFromFile`. To change the graphics world for the new movie, your application should use the `SetMovieGWorld` function, which is described on page 2-159.

SPECIAL CONSIDERATIONS

The Movie Toolbox automatically sets the movie's graphics world based upon the current graphics port. Be sure that your application's graphics world is valid before you call this function.

ERROR CODES

<code>badImageDescription</code>	-2001	Problem with an image description
<code>badPublicMovieAtom</code>	-2002	Movie file corrupted
<code>cantFindHandler</code>	-2003	Cannot locate a handler
<code>cantOpenHandler</code>	-2004	Cannot open a handler

File Manager errors

Memory Manager errors

Resource Manager errors

NewMovieFromHandle

The `NewMovieFromHandle` function creates a movie in memory from a movie resource or a handle you obtained from the `PutMovieIntoHandle` function.

```
pascal OSErr NewMovieFromHandle (Movie *theMovie, Handle h,
                                short newMovieFlags,
                                Boolean *dataRefWasChanged);
```

theMovie Contains a pointer to a field that is to receive the new movie's identifier. If the function cannot load the movie, the returned identifier is set to `nil`.

h Contains a handle to the movie resource from which the movie is to be loaded.

newMovieFlags

Controls the operation of the `NewMovieFromHandle` function. The following flags are available (be sure to set unused flags to 0):

newMovieActive

Controls whether the new movie is active. Set this flag to 1 to make the new movie active. You can make a movie active or inactive by calling the `SetMovieActive` function, which is described on page 2-145.

newMovieDontResolveDataRefs

Controls how completely the Movie Toolbox resolves data references in the movie resource. If you set this flag to 0, the toolbox tries to completely resolve all data references in the resource. This may involve searching for files on remote volumes. If you set this flag to 1, the Movie Toolbox only looks in the specified file.

If the Movie Toolbox cannot completely resolve all the data references, it still returns a valid movie identifier. In this case, the Movie Toolbox also sets the current error value to `couldNotResolveDataRef`.

newMovieDontAskUnresolvedDataRefs

Controls whether the Movie Toolbox asks the user to locate files. If you set this flag to 0, the Movie Toolbox asks the user to locate files that it cannot find on available volumes. If the Movie Toolbox cannot locate a file even with the user's help, the function returns a valid movie identifier and sets the current error value to `couldNotResolveDataRef`.

newMovieDontAutoAlternate

Controls whether the Movie Toolbox automatically selects enabled tracks from alternate track groups. If you set this flag to 1, the Movie Toolbox does not automatically select tracks for the movie—you must enable tracks yourself.

dataRefWasChanged

Contains a pointer to a Boolean value. The Movie Toolbox sets the Boolean value to indicate whether it had to change any data references in order to resolve them. The toolbox sets the Boolean value to `true` if any references were changed. Set the `dataRefWasChanged` parameter to `nil` if you do not want to receive this information.

DESCRIPTION

The `NewMovieFromHandle` function returns the new movie's identifier. If the function cannot create the movie, the function sets the returned identifier to `nil`.

Your application can use the `NewMovieFromFile` function, described in the previous section, to load a movie from a movie file that was opened with the `OpenMovieFile` function. If you are loading a movie from a resource, use the `NewMovieFromResource` function instead. The Movie Toolbox uses information about the resource file when it resolves data references in the movie.

The Movie Toolbox sets many movie characteristics to default values. If you want to change these defaults, your application must call other Movie Toolbox functions. For example, the Movie Toolbox sets the movie's graphics world to the one that is active when you call `NewMovieFromHandle`. To change the graphics world for the new movie, your application should use the `SetMovieGWorld` function, which is described on page 2-159.

Movie Toolbox

SPECIAL CONSIDERATIONS

The Movie Toolbox automatically sets the movie's graphics world based upon the current graphics port. Be sure that your application's graphics world is valid before you call this function.

ERROR CODES

<code>badImageDescription</code>	-2001	Problem with an image description
<code>badPublicMovieAtom</code>	-2002	Movie file corrupted
<code>cantFindHandler</code>	-2003	Cannot locate a handler
<code>cantOpenHandler</code>	-2004	Cannot open a handler

File Manager errors

Memory Manager errors

Resource Manager errors

NewMovie

The `NewMovie` function creates a new movie in memory. The Movie Toolbox initializes the data structures for the new movie, which contains no tracks. Your application assigns the data to the movie by calling the functions that are described later in "Creating Tracks and Media Structures" beginning on page 2-150.

```
pascal Movie NewMovie (long newMovieFlags);
```

`newMovieFlags`

Specifies control information for the new movie. The following flags are available (be sure to set unused flags to 0):

`newMovieActive`

Controls whether the new movie is active. Set this flag to 1 to make the new movie active. A movie that does not have any tracks can still be active. When the Movie Toolbox tries to play the movie, no images are displayed, because there is no movie data. You can make a movie active or inactive by calling the `SetMovieActive` function, which is described on page 2-145.

`newMovieDontAutoAlternate`

Controls whether the Movie Toolbox automatically selects enabled tracks from alternate track groups. If you set this flag to 1, the Movie Toolbox does not automatically select tracks for the movie—you must enable tracks yourself.

DESCRIPTION

The `NewMovie` function returns the identifier for the new movie. If the function fails, the returned identifier is set to `nil`. Use the `GetMoviesError` function (described on page 2-85) to obtain the result code.

Movie Toolbox

The Movie Toolbox sets many movie characteristics to default values. If you want to change these defaults, your application must call other Movie Toolbox functions. For example, the Movie Toolbox sets the movie’s graphics world to the one that is active when you call `NewMovie`. To change the graphics world for the new movie, your application should use the `SetMovieGWorld` function, which is described on page 2-159.

The default QuickTime movie time scale is 600 units per second; however, this number may change in the future. The default time scale was chosen because it is convenient for working with common video frame rates of 30, 25, 24, 15, 12, 10, and 8.

You should use the `NewMovie` function only if you have not created a new movie and movie file by calling the `CreateMovieFile` function.

▲ WARNING
The Movie Toolbox automatically sets the movie’s graphics world based upon the current graphics port. Be sure that your application’s graphics port is valid before you call this function. ▲

ERROR CODES

<code>movieToolboxUninitialized</code>	–2020	You haven’t initialized the Movie Toolbox
Memory Manager errors		

ConvertFileToMovieFile

The `ConvertFileToMovieFile` takes a specified file and converts it to a movie file.

```
pascal OSErr ConvertFileToMovieFile (const FSSpec *inputFile,
                                     const FSSpec *outputFile,
                                     OSType creator,
                                     ScriptCode scriptTag,
                                     short *resID, long flags,
                                     ComponentInstance userComp,
                                     MovieProgressProcPtr proc,
                                     long refCon);
```

<code>inputFile</code>	Contains a pointer to the file system specification for the file to be converted into a movie file.
<code>outputFile</code>	Contains a pointer to the file specification for the destination movie file.
<code>creator</code>	Specifies the creator value for the file if it is a new one.

Movie Toolbox

<code>scriptTag</code>	Specifies the script in which the movie file should be converted. Use the <code>smSystemScript</code> constant to use the system script; use the <code>smCurrentScript</code> constant to use the current script. See <i>Inside Macintosh: Text</i> for more information about scripts and script tags.
<code>resID</code>	Contains a pointer to a field that is to receive the resource ID of the file to be converted. If you don't want to receive the resource ID, set this parameter to <code>nil</code> .
<code>flags</code>	Controls movie file conversion flags. The following value is valid: <div style="margin-left: 40px;"><code>createMovieFileDeleteCurFile</code> 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 <code>fileSpec</code> parameter already exists, the Movie Toolbox uses the existing file. In this case, the toolbox ensures that the file has both a data and a resource fork.</div>
<code>userComp</code>	Indicates a component or component instance of the movie export component you want to perform the conversion. Otherwise, set this parameter to 0 for the Movie Toolbox to choose the appropriate component. If you pass in a component instance, it will be used by <code>ConvertFileToMovieFile</code> . This allows you to communicate directly with the component before using this function to establish any conversion parameters. If you pass in a component ID, an instance is created and closed within this function. For details on movie export components, see <i>Inside Macintosh: QuickTime Components</i> .
<code>proc</code>	Points to your progress function. To remove a movie's progress function, set this parameter to <code>nil</code> . Set this parameter to -1 for the Movie Toolbox to provide a default progress function. See "Progress Functions," which begins on page 2-354, for the interface your progress function must support.
<code>refCon</code>	Specifies a reference constant. The Movie Toolbox passes this value to your progress function.

DESCRIPTION

Because some conversions may take a nontrivial amount of time, you can pass a standard movie progress function in the `proc` and `refCon` parameters.

ConvertMovieToFile

The `ConvertMovieToFile` function takes a specified movie (or a single track within that movie) and converts it into a specified file and type.

```
pascal OSErr ConvertMovieToFile(Movie theMovie, Track onlyTrack,
                                const FSSpec *outputFile,
                                OSType fileType, OSType creator,
                                ScriptCode scriptTag,
                                short *resID, long flags,
                                ComponentInstance userComp);
```

<code>theMovie</code>	Specifies the source movie for this conversion operation. Your application obtains this movie identifier from such functions as <code>NewMovie</code> , <code>NewMovieFromFile</code> , and <code>NewMovieFromHandle</code> (described on page 2-92, page 2-88, and page 2-90, respectively).
<code>onlyTrack</code>	Specifies the track within the source movie for this conversion operation. To specify all tracks, set the value of this parameter to 0.
<code>outputFile</code>	Contains a pointer to the file specification for the destination file.
<code>fileType</code>	Specifies the data type of the destination file for the movie specified in the parameter <code>theMovie</code> .
<code>creator</code>	Specifies the creator value for the output file if it is a new one.
<code>scriptTag</code>	Specifies the script into which the movie should be converted if the output file is a new one. Use the Script Manager constant <code>smSystemScript</code> to use the system script; use the <code>smCurrentScript</code> constant to use the current script. See <i>Inside Macintosh: Text</i> for more information about scripts and script tags.
<code>resID</code>	Contains a pointer to a field that is to receive the resource ID of the open movie. If you don't want to receive this information, set the <code>resID</code> parameter to <code>nil</code> .
<code>flags</code>	Set this parameter to 0.
<code>userComp</code>	If you want a particular movie export component to perform the conversion, you may pass the component or an instance of that component in this parameter. Otherwise, set it to 0 to allow the Movie Toolbox to use the appropriate component. If you pass in a component instance, it is used by <code>ConvertMovieToFile</code> . This allows you to communicate directly with the component before making this call to establish any conversion parameters. If you pass in a component ID, an instance is created and closed within this call.

DisposeMovie

The `DisposeMovie` function frees any memory being used by a movie, including the memory used by the movie's tracks and media structures. Your application should call this function when it is done working with a movie.

```
pascal void DisposeMovie (Movie theMovie);
```

theMovie Identifies the movie to be freed. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, or `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

SPECIAL CONSIDERATIONS

Do not dispose of a movie if it has any special clients—for example, if it has an attached movie controller component. Only dispose of the movie after any clients are done with it.

ERROR CODES

`invalidMovie` -2010 This movie is corrupted or invalid

CreateMovieFile

The `CreateMovieFile` function creates an open movie file, opens the movie file, creates an empty movie which references the file, and opens the movie file with write permission.

```
pascal OSErr CreateMovieFile (const FSSpec *fileSpec,
                              OSType creator,
                              ScriptCode scriptTag,
                              long createMovieFileFlags,
                              short *resRefNum,
                              Movie *newMovie);
```

fileSpec Contains a pointer to the file system specification for the movie file to be created.

creator Specifies the creator value for the new file.

scriptTag Specifies the script in which the movie file should be created. Use the `smSystemScript` constant to use the system script; use the `smCurrentScript` constant to use the current script. See *Inside Macintosh: Text* for more information about scripts and script tags.

Movie Toolbox

`createMovieFileFlags`

Controls movie-file creation flags. The following flags are available:

`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 creating the new movie file. If you set this flag to 0 and the file specified by the `fileSpec` parameter already exists, the Movie Toolbox uses the existing file. In this case, the toolbox ensures that the file has both a data and a resource fork.

`createMovieFileDontCreateMovie`

Controls whether the `CreateMovieFile` function creates a new movie in the movie file. If you set this flag to 1, the Movie Toolbox does not create a movie in the new movie file. In this case, the function ignores the `newMovie` parameter. If you set this flag to 0, the Movie Toolbox creates a movie and returns the movie identifier in the field referred to by the `newMovie` parameter.

`createMovieFileDontOpenFile`

Controls whether the `CreateMovieFile` function opens the new movie file. If you set this flag to 1, the Movie Toolbox does not open the new movie file. In this case, the function ignores the `resRefNum` parameter. If you set this flag to 0, the Movie Toolbox opens the new movie file and returns its reference number into the field referred to by the `resRefNum` parameter.

`newMovieActive`

Controls whether the new movie is active. Set this flag to 1 to make the new movie active. A movie that does not have any tracks can still be active. When the Movie Toolbox tries to play the movie, no images are displayed, because there is no movie data. You can make a movie active or inactive by calling the `SetMovieActive` function, which is described on page 2-145.

`newMovieDontAutoAlternate`

Controls whether the Movie Toolbox automatically selects enabled tracks from alternate track groups. If you set this flag to 1, the Movie Toolbox does not automatically select tracks for the movie—you must enable tracks yourself.

`resRefNum`

Contains a pointer to a field that is to receive the file reference number for the opened movie file. Your application must use this value when calling other Movie Toolbox functions that work with movie files. If you set this parameter to `nil`, the Movie Toolbox creates the movie file but does not open the file.

Movie Toolbox

newMovie Contains a pointer to a field that is to receive the identifier of the new movie. The `CreateMovieFile` function returns the identifier of the new movie. If the function could not create a new movie, it sets this returned value to `nil`. If you set this parameter to `nil`, the Movie Toolbox does not create a movie.

ERROR CODES

`movieToolboxUninitialized` -2020 You haven't initialized the Movie Toolbox

File Manager errors

Memory Manager errors

SEE ALSO

You can delete a movie file by calling the `DeleteMovieFile` function, which is described on page 2-100.

Your application can use the functions described in "Creating Tracks and Media Structures," which begins on page 2-150, to place movie data into the new movie file.

OpenMovieFile

The `OpenMovieFile` function opens a specified movie file. Your application identifies the movie file with a file system specification.

```
pascal OSErr OpenMovieFile (const FSSpec *fileSpec,
                             short *resRefNum, char perms);
```

fileSpec Contains a pointer to the file system specification for the movie file to be opened.

resRefNum Contains a pointer to a field that is to receive the file reference number for the opened movie file. Your application must use this value when calling other Movie Toolbox functions that work with movie files. This reference number refers to the file fork that contains the movie resource—if the movie is stored in the data fork of the file, the returned reference number corresponds to the data fork.

perms Specifies the permission level for the file. If your application is only going to play the movie that is stored in the file, you can open the file with read permission. If you plan to add data to the file or change data in the file, you should open the file with write permission. Supply a valid File Manager permission value. See *Inside Macintosh: Files* for valid values.

DESCRIPTION

Your application must open a movie file before reading movie data from it or writing movie data to it. You can open a movie file more than once—be sure to call `CloseMovieFile` (described in the next section) once for each time you call `OpenMovieFile`.

Note that opening the movie file with write permission does not prevent other applications from reading data from the movie file.

If the specified file has a resource fork, the `OpenMovieFile` function opens the resource fork and returns a file reference number to the resource fork. If the movie file does not have a resource fork (that is, it is a single-fork movie file—see the chapter “Movie Resource Formats” in this book for more information), the `OpenMovieFile` function opens the data fork instead. In this case, your application cannot use the `AddMovieResource` function (described on page 2-102) with the movie file.

ERROR CODES

<code>movieToolboxUninitialized</code>	-2020	You haven’t initialized the Movie Toolbox
--	-------	---

File Manager errors
Memory Manager errors

CloseMovieFile

The `CloseMovieFile` function closes an open movie file.

```
pascal OSErr CloseMovieFile (short resRefNum);
```

`resRefNum` Specifies the movie file to close. Your application obtains this reference number from the `OpenMovieFile` function, which is described in the previous section.

DESCRIPTION

Your application should call this function when you are done working with a movie file. You must call this function once each time you open a movie file. You can still use the movie. If you are not editing the movie, it is advisable to close it.

ERROR CODES

File Manager errors

DeleteMovieFile

The `DeleteMovieFile` function deletes a movie file.

```
pascal OSErr DeleteMovieFile (const FSSpec *fileSpec);
```

`fileSpec` Contains a pointer to the file system specification for the movie file to be deleted.

DESCRIPTION

Do not use the file system to delete movie files. The Movie Toolbox maintains references between files.

ERROR CODES

File Manager errors

Saving Movies

The Movie Toolbox provides a set of high-level functions for storing movies within files. These files have a file type of 'Moov' and a resource type of 'moov'. Your application can gain access to existing movies with either the `NewMovieFromFile` function or the `NewMovieFromDataFork` function (described on page 2-88 and page 2-109, respectively). Once you have loaded the movie, your application uses the functions that are described in this section to save any changes you have made to the movie.

You can use the `AddMovieResource` function to add a new movie resource to a movie file. Your application can use this function to save a movie that it created using the functions described in "Functions for Editing Movies" beginning on page 2-242. You can use the `UpdateMovieResource` function to replace an existing movie resource in a movie file. You can remove a movie resource by calling the `RemoveMovieResource` function.

The movie resources that your application creates with the `AddMovieResource` and `UpdateMovieResource` functions may contain references to movie data. These references identify the data that constitute the movie. However, the movie data can be stored outside of the movie file. If you want to create a movie file that contains all of its movie data, use the `FlattenMovie` function. If you want to create a single-fork movie file, use the `FlattenMovieData` function.

The `PutMovieIntoHandle` function places a QuickTime movie into a handle. You can then convert the movie into specialized data formats.

The `HasMovieChanged` and `ClearMovieChanged` functions allow your application to work with the movie changed flag that is maintained by the Movie Toolbox. You can use this flag to determine whether a movie has been changed.

The movie changed flag indicates whether you have changed the movie. Such actions as editing the movie, adding samples to a media, or changing a data reference cause the flag to indicate that the movie has changed. There are several operations that the movie changed flag does not reflect, including changing the volume, rate, or time settings for the movie. These settings change frequently when a movie is played. Your application must monitor these settings itself.

The Movie Toolbox also supplies functions for storing and retrieving movies that are stored in the data fork of a file. These functions provide robust data reference resolution and improve low memory performance. The `NewMovieFromDataFork` function enables you to retrieve a movie that is stored anywhere in the data fork of a file. You can use the `PutMovieIntoDataFork` function to store an atom version of a specified movie in the data fork of a file.

HasMovieChanged

The `HasMovieChanged` function allows your application to determine whether a movie has changed and needs to be saved.

```
pascal Boolean HasMovieChanged (Movie theMovie);
```

theMovie Specifies the movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

DESCRIPTION

The `HasMovieChanged` function returns a Boolean value that reflects the contents of the movie changed flag. The function sets the returned value to `true` if the movie has been changed in such a way that it should be saved. Otherwise, the returned value is set to `false`.

Your application can clear the movie changed flag, indicating that the movie has not changed, by calling the `ClearMovieChanged` function, which is described in the next section.

ERROR CODES

<code>invalidMovie</code>	-2010	This movie is corrupted or invalid
---------------------------	-------	------------------------------------

SEE ALSO

Both the `AddMovieResource` function (described on page 2-102) and the `UpdateMovieResource` function (described on page 2-103) update the movie file and clear the movie changed flag, indicating that the movie has not been changed.

ClearMovieChanged

The `ClearMovieChanged` function sets the movie changed flag to indicate that the movie has not been changed.

```
pascal void ClearMovieChanged (Movie theMovie);
```

theMovie Specifies the movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

ERROR CODES

`invalidMovie` -2010 This movie is corrupted or invalid

SEE ALSO

Your application can read the contents of the movie changed flag by calling the `HasMovieChanged` function, which is described in the previous section. Both the `AddMovieResource` and `UpdateMovieResource` functions also clear the movie changed flag.

AddMovieResource

The `AddMovieResource` function adds a movie resource to a specified resource file. Your application identifies the movie to be added to the movie file.

```
pascal OSErr AddMovieResource (Movie theMovie, short resRefNum,
                                short *resId,
                                const StringPtr resName);
```

theMovie Specifies the movie you wish to add to the movie file. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

resRefNum Identifies the movie file to which the resource is to be added. Your application obtains this value from the `OpenMovieFile` function, described on page 2-98. The movie file specified by this parameter cannot be a single-fork movie file.

resId Contains a pointer to a field that contains the resource ID number for the new resource. If the field referred to by the `resId` parameter is set to 0, the Movie Toolbox assigns a unique resource ID number to the new resource. The toolbox then returns the movie's resource ID number in the field referred to by the `resId` parameter. The `AddMovieResource`

function assigns resource ID numbers sequentially, starting at 128. If the `resId` parameter is set to `nil`, the Movie Toolbox assigns a unique resource ID number to the new resource and does not return that resource's ID value.

`resName` Points to a character string that contains the name of the movie resource. If you set the `resName` parameter to `nil`, the toolbox creates an unnamed resource.

DESCRIPTION

The `AddMovieResource` function adds the movie to the file, effectively saving any changes you have made to the movie. This function does not work with single-fork movie files.

After updating the movie file, `AddMovieResource` clears the movie changed flag, indicating that the movie has not been changed.

ERROR CODES

`invalidMovie` -2010 This movie is corrupted or invalid

File Manager errors
Memory Manager errors
Resource Manager errors

UpdateMovieResource

The `UpdateMovieResource` function replaces the contents of a movie resource in a specified movie file. You specify the movie that is to be placed into the resource.

This function can accommodate single-fork movie files.

```
pascal OSErr UpdateMovieResource (Movie theMovie, short resRefNum,
                                   short resId,
                                   const StringPtr resName);
```

`theMovie` Specifies the movie you wish to place in the movie file. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

`resRefNum` Identifies the movie file that contains the resource to be changed. Your application obtains this value from the `OpenMovieFile` function, described on page 2-98. If this parameter specifies a single-fork movie file using the `movieInDataForResID(-1)` constant, the Movie Toolbox places the movie resource into the file's data fork.

`resId` Specifies the resource to be changed.

Movie Toolbox

`resName` Points to a new name for the resource. If you do not want to change the resource's name, set this parameter to `nil`.

DESCRIPTION

After updating the movie file, the `UpdateMovieResource` function clears the movie changed flag, indicating that the movie has not been changed.

ERROR CODES

`invalidMovie` -2010 This movie is corrupted or invalid

File Manager errors
Memory Manager errors
Resource Manager errors

RemoveMovieResource

The `RemoveMovieResource` function removes a movie resource from a specified movie file.

```
pascal OSErr RemoveMovieResource (short resRefNum, short resId);
```

`resRefNum` Identifies the movie file that contains the movie resource. Your application obtains this value from the `OpenMovieFile` function, described on page 2-98.

`resId` Specifies the resource to be removed.

ERROR CODES

File Manager errors
Resource Manager errors

PutMovieIntoHandle

The `PutMovieIntoHandle` function creates a new movie resource for you. You can use this handle to store a QuickTime movie in a specialized storage format.

```
pascal OSErr PutMovieIntoHandle (Movie theMovie,
                                Handle publicMovie);
```

Movie Toolbox

`theMovie` Specifies the movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

`publicMovie` Contains the handle that is to receive the new movie resource. The `PutMovieIntoHandle` function places the new movie resource into this handle. The function resizes the handle if necessary.

DESCRIPTION

Note that you cannot use this new movie with other Movie Toolbox functions, except for the `NewMovieFromHandle` function. You can use the `NewMovieFromHandle` function, described on page 2-90, to load a movie from a handle.

SPECIAL CONSIDERATIONS

Movies saved using `PutMovieIntoHandle` contain less robust data references than those created using the `AddMovieResource` or `PutMovieIntoDataFork` functions (described on page 2-102 and page 2-110, respectively).

ERROR CODES

`invalidMovie` -2010 This movie is corrupted or invalid

Memory Manager errors

FlattenMovie

The `FlattenMovie` function creates a new movie file containing a specified movie. This file also contains all the data for the movie—that is, the Movie Toolbox resolves any data references and includes the corresponding movie data in the new movie file.

```
pascal void FlattenMovie (Movie theMovie, long movieFlattenFlags,
                          const FSSpec *theFile,
                          OSType creator, ScriptCode scriptTag,
                          long createMovieFileFlags,
                          short *resId, const StringPtr resName);
```

`theMovie` Specifies the movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

Movie Toolbox

`movieFlattenFlags`

Controls the process of adding movie data to the new movie file. The following flags are available (be sure to set unused flags to 0):

`flattenAddMovieToDataFork`

Causes the movie to be placed in the data fork of the new movie file, as well as in the resource fork. You may use this flag to create movie files that are more easily moved to other computer systems from your Macintosh.

`flattenDontInterleaveFlatten`

Allows you to disable the Movie Toolbox's data storage optimizations. By default, the Movie Toolbox stores movie data in a format that is optimized for playback. Set this flag to 1 to disable these optimizations.

`flattenActiveTracksOnly`

Causes the Movie Toolbox to add only enabled movie tracks to the new movie file. You can use the `SetTrackEnabled` function, described on page 2-147, to enable and disable movie tracks.

`theFile` Contains a pointer to the file system specification for the movie file to be created.

`creator` Specifies the creator value for the new file.

`scriptTag` Specifies the script in which the movie file should be created. Set this parameter to the Script Manager constant `smSystemScript` to use the system script; set it to `smCurrentScript` to use the current script. See *Inside Macintosh: Text* for more information about scripts and script tags.

`createMovieFileFlags`

Controls file creation options. The following flag is available:

`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 creating the new movie file. If this flag is set to 0 and the file specified by the `fileSpec` parameter already exists, the Movie Toolbox uses the existing file. In this case, the toolbox ensures that the file has both a data and a resource fork. If this flag is not set, the data is appended to the file.

`resId` Contains a pointer to a field that contains the resource ID number for the new resource. If the field referred to by the `resId` parameter is set to 0, the Movie Toolbox assigns a unique resource ID number to the new resource. The toolbox then returns the movie's resource ID number in the field referred to by the `resId` parameter. The Movie Toolbox assigns resource ID numbers sequentially, starting at 128. If the `resId` parameter is set to `nil`, the Movie Toolbox assigns a unique resource ID number to the new resource and does not return that resource's ID value.

`resName` Points to a character string with the name of the movie resource. If you set the `resName` parameter to `nil`, the toolbox creates an unnamed resource.

DESCRIPTION

The toolbox places the movie resource into the resource fork of the movie file. The Movie Toolbox does not alter the source movie.

The Movie Toolbox calls your progress function during long operations.

ERROR CODES

invalidMovie	-2010	This movie is corrupted or invalid
progressProcAborted	-2019	Your progress function returned an error
cantCreateSingleForkFile	-2022	Error trying to create a single-fork file
File Manager errors		
Memory Manager errors		
Resource Manager errors		

FlattenMovieData

The FlattenMovieData function creates a new movie file and creates a new movie that contains all of its movie data. However, unlike the FlattenMovie function described in the previous section, this function does not add the new movie resource to the new movie file. Instead, the FlattenMovieData function returns the new movie to your application. Your application must dispose of the returned movie.

```
pascal Movie FlattenMovieData (Movie theMovie,
                                long movieFlattenFlags,
                                const FSSpec *theFile,
                                OSType creator,
                                ScriptCode scriptTag,
                                long createMovieFileFlags);
```

theMovie Specifies the movie for this operation. Your application obtains this movie identifier from such functions as NewMovie, NewMovieFromFile, and NewMovieFromHandle (described on page 2-92, page 2-88, and page 2-90, respectively).

movieFlattenFlags Controls the process of adding movie data to the new movie file. These flags affect how the toolbox adds movies to the new movie file later. The following flags are available (be sure to set unused flags to 0):

flattenAddMovieToDataFork Causes the movie to be placed in the data fork of the new movie file. You may use this flag to create single-fork movie files, which can be more easily moved to other computer systems from your Macintosh.

Movie Toolbox

<code>flattenDontInterleaveFlatten</code>	Allows you to disable the Movie Toolbox's data storage optimizations. By default, the Movie Toolbox stores movie data in a format that is optimized for the storage device. Set this flag to 1 to disable these optimizations.
<code>flattenActiveTracksOnly</code>	Causes the Movie Toolbox to add only enabled movie tracks to the new movie file. You can use the <code>SetTrackEnabled</code> function, which is described on page 2-147, to enable and disable movie tracks.
<code>theFile</code>	Contains a pointer to the file system specification for the movie file to be created.
<code>creator</code>	Specifies the creator value for the new file.
<code>scriptTag</code>	Specifies the script in which the movie file should be created. Set this parameter to <code>smSystemScript</code> to use the system script; set it to <code>smCurrentScript</code> to use the current script. See <i>Inside Macintosh: Text</i> for more information about scripts and script tags.
<code>creationFlags</code>	Controls file creation options. The following flag is available:
<code>createMovieFileDeleteCurFile</code>	Indicates whether to delete an existing file. If you set this flag to 1, the Movie Toolbox deletes the file (if it exists) before creating the new movie file. If this flag is set to 0 and the file specified by the <code>fileSpec</code> parameter already exists, the Movie Toolbox uses the existing file. In this case, the toolbox ensures that the file has both a data and a resource fork. If this flag isn't set, the data is appended to the file.

DESCRIPTION

The `FlattenMovieData` function returns the movie identifier of the new movie. If the function could not create the movie, it sets this returned identifier to `nil`.

You can also use this function to create a single-fork movie file. Set the `flattenAddMovieToDataFork` flag in the `movieFlattenFlags` parameter to 1. The Movie Toolbox then places the movie into the data fork of the movie file.

The Movie Toolbox calls your progress function during long operations.

The Movie Toolbox does not alter the source movie.

ERROR CODES

<code>invalidMovie</code>	-2010	This movie is corrupted or invalid
<code>progressProcAborted</code>	-2019	Your progress function returned an error
<code>cantCreateSingleForkFile</code>	-2022	Error trying to create a single-fork file

File Manager errors

Memory Manager errors

NewMovieFromDataFork

The `NewMovieFromDataFork` function enables you to retrieve a movie that is stored anywhere in the data fork of a specified file.

```
pascal OSErr NewMovieFromDataFork (Movie *theMovie,
                                   short fRefNum,
                                   long fileOffset,
                                   short newMovieFlags,
                                   Boolean *dataRefWasChanged);
```

<code>theMovie</code>	Contains a pointer to the movie identifier for the movie to be retrieved. Your application obtains this movie identifier from such functions as <code>NewMovie</code> , <code>NewMovieFromFile</code> , and <code>NewMovieFromHandle</code> (described on page 2-92, page 2-88, and page 2-90, respectively).
<code>fRefNum</code>	Contains a file reference number to a file that is already open.
<code>fileOffset</code>	Specifies the starting file offset of the atom in the data fork of the file specified by the <code>fRefNum</code> parameter.
<code>newMovieFlags</code>	Contains the standard flags in the <code>newMovie</code> enumeration.
<code>newMovieActive</code>	Controls whether the new movie is active. Set this flag to 1 to make the new movie active. A movie that does not have any tracks can still be active. When the Movie Toolbox tries to play the movie, no images are displayed, because there is no movie data. You can make a movie active or inactive by calling the <code>SetMovieActive</code> function, which is described on page 2-145.
<code>newMovieDontAutoAlternate</code>	Controls whether the Movie Toolbox automatically selects enabled tracks from alternate track groups. If you set this flag to 1, the Movie Toolbox does not automatically select tracks for the movie—you must enable tracks yourself.
<code>newMovieDontResolveDataRefs</code>	Controls how completely the Movie Toolbox resolves data references in the movie resource. If you set this flag to 0, the toolbox tries to completely resolve all data references in the resource. This may involve searching for files on remote volumes. If you set this flag to 1, the Movie Toolbox only looks in the specified file. If the Movie Toolbox cannot completely resolve all the data references, it still returns a valid movie identifier. In this case, the Movie Toolbox also sets the current error value to <code>couldNotResolveDataRef</code> .

Movie Toolbox

`newMovieDontAskUnresolvedDataRefs`

Controls whether the Movie Toolbox asks the user to locate files. If you set this flag to 0, the Movie Toolbox asks the user to locate files that it cannot find on available volumes. If the Movie Toolbox cannot locate a file even with the user's help, the function returns a valid movie identifier and sets the current error value to `couldNotResolveDataRef`.

`dataRefWasChanged`

Contains a pointer to a Boolean value. The Movie Toolbox sets the Boolean to indicate whether it had to change any data references while resolving them. The toolbox sets the Boolean value to `true` if any references were changed. Use the `UpdateMovieResource` function (described on page 2-103) to preserve these changes.

Set the `dataRefWasChanged` parameter to `nil` if you do not want to receive this information. See the "Creating Tracks and Media Structures" beginning on page 2-150 for more information about data references.

ERROR CODES

<code>badImageDescription</code>	-2001	Problem with an image description
<code>badPublicMovieAtom</code>	-2002	Movie file corrupted
<code>cantFindHandler</code>	-2003	Cannot locate a handler
<code>cantOpenHandler</code>	-2004	Cannot open a handler
File Manager errors		
Memory Manager errors		

PutMovieIntoDataFork

The `PutMovieIntoDataFork` function allows you to store a movie in the data fork of a given file.

```
pascal OSErr PutMovieIntoDataFork (Movie theMovie, short fRefNum,
                                     long offset, long maxSize);
```

<code>theMovie</code>	Identifies the movie to be stored in the data fork of an atom. Your application obtains this movie identifier from such functions as <code>NewMovie</code> , <code>NewMovieFromFile</code> , and <code>NewMovieFromHandle</code> (described on page 2-92, page 2-88, and page 2-90, respectively).
<code>fRefNum</code>	Contains a file reference number for the data fork of the given file. You pass in an open write path in the <code>fRefNum</code> parameter.
<code>offset</code>	Indicates where the movie should be written.
<code>maxSize</code>	Indicates the largest number of bytes that may be written.

DESCRIPTION

If necessary, the file will be extended. If there is insufficient space to write the movie, either due to a lack of disk space or because of the limit specified in the `maxSize` parameter, this function returns a `dskFullErr` error code. If there is no limit on how much space the movie may take up in the file, pass 0 in the `maxSize` parameter.

ERROR CODES

`invalidMovie` -2010 This movie is corrupted or invalid
Memory Manager errors
File Manager errors

Controlling Movie Playback

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 "Working with Movie Time" beginning on page 2-184.

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. These advanced functions are described in "Functions That Modify Movie Properties" beginning on page 2-157.

StartMovie

The `StartMovie` function starts the movie playing from the current movie time, which is where the movie last stopped playing. Before playing the movie, the Movie Toolbox makes the movie active, prerolls the movie, and sets the movie to its preferred playback rate. You can use the `SetMoviePreferredRate` function (described on page 2-130) to change this setting.

```
pascal void StartMovie (Movie theMovie);
```

`theMovie` Specifies the movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

Movie Toolbox

DESCRIPTION

Note that a movie's current time is saved when a movie is stored in a movie file. Therefore, your application should appropriately position a movie before playing the movie—use the `GoToBeginningOfMovie` function (described on page 2-113) to set a movie to play from its start.

You are not required to call `StartMovie` to start a movie. This function is included merely for convenience.

ERROR CODES

`invalidMovie` -2010 This movie is corrupted or invalid
Memory Manager errors

SEE ALSO

You can also start a movie playing by calling the `SetMovieRate` function (described on page 2-187) and setting the movie's rate to a nonzero value.

StopMovie

The `StopMovie` function stops the playback of a movie.

```
pascal void StopMovie (Movie theMovie);
```

`theMovie` Specifies the movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

ERROR CODES

`invalidMovie` -2010 This movie is corrupted or invalid

SEE ALSO

You can use the `StartMovie` function described in the previous section to resume playing.

GoToBeginningOfMovie

The `GoToBeginningOfMovie` function repositions a movie to play from its start.

```
pascal void GoToBeginningOfMovie (Movie theMovie);
```

theMovie Specifies the movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

DESCRIPTION

If you have defined an active movie segment, the `GoToBeginningOfMovie` function repositions to the start of the active segment. The **active movie segment** is the part of the movie that your application is interested in playing. By default, the active movie segment is set to be the entire movie. You may wish to change this to be some segment of the movie—for example, if you wish to play a user’s selection repeatedly. By setting the active movie segment, you guarantee that the Movie Toolbox uses no samples from outside of that range while playing the movie.

If the movie is in preview mode, the function goes to the start of the preview segment of the movie. In all other cases, this function moves you to the start of the movie, where the movie time value is 0.

SPECIAL CONSIDERATIONS

Movies need not be at the start position when they are saved. The Movie Toolbox stores a movie’s time position in the movie when it is saved. If you want to play a movie from the beginning, your application should call the `GoToBeginningOfMovie` function before playing a movie you have loaded from a movie file.

ERROR CODES

`invalidMovie` -2010 This movie is corrupted or invalid

SEE ALSO

You can use the `SetMovieActiveSegment` and `GetMovieActiveSegment` functions to work with the active segment. For details, see “Enhancing Movie Playback Performance” beginning on page 2-134.

GoToEndOfMovie

The `GoToEndOfMovie` function repositions a movie to play from its end.

```
pascal void GoToEndOfMovie (Movie theMovie);
```

theMovie Specifies the movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

DESCRIPTION

If you have defined an active movie segment, the `GoToEndOfMovie` function repositions the movie to the end of the active segment. If the movie is in preview mode, the function goes to the end of the preview segment of the movie. In all other cases, this function moves you to the end of the movie.

ERROR CODES

`invalidMovie` -2010 This movie is corrupted or invalid

SEE ALSO

You can use the `SetMovieActiveSegment` and `GetMovieActiveSegment` functions to work with the active segment. For details, see “Enhancing Movie Playback Performance” beginning on page 2-134.

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. The preview is intended to give the user an idea of a movie’s contents.

A movie poster is a still frame representing the movie.

This section describes the Movie Toolbox functions that 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 to control movie playback—these functions are described on page 2-111 and page 2-112, respectively. 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.

SetTrackUsage

The `SetTrackUsage` function allows your application to specify whether a track is used in a movie, its preview, its poster, or a combination of these.

```
pascal void SetTrackUsage (Track theTrack, long usage);
```

theTrack Specifies the track for this operation. Your application obtains this track identifier from such Movie Toolbox functions as `NewMovieTrack` and `GetMovieTrack` (described on page 2-151 and page 2-204, respectively).

usage Contains flags that specify how the track is to be used. The following flags are defined (be sure to set unused flags to 0):

- `trackUsageInMovie`
The track is used in the movie. If this flag is set to 1, the track is used in the movie.
- `trackUsageInPreview`
The track is used in the preview. If this flag is set to 1, the track is used in the preview.
- `trackUsageInPoster`
The track is used in the poster. If this flag is set to 1, the track is used in the poster.

ERROR CODES

<code>invalidTrack</code>	-2009	This track is corrupted or invalid
---------------------------	-------	------------------------------------

SEE ALSO

Your application can determine how a track is used by calling the `GetTrackUsage` function, which is described in the next section.

GetTrackUsage

The `GetTrackUsage` function allows your application to determine whether a track is used in a movie, its preview, its poster, or a combination of these. Your application can specify how a track is used by calling the `SetTrackUsage` function, which is described in the previous section.

```
pascal long GetTrackUsage (Track theTrack);
```

theTrack Specifies the track for this operation. Your application obtains this track identifier from such Movie Toolbox functions as `NewMovieTrack` and `GetMovieTrack` (described on page 2-151 and page 2-204, respectively).

DESCRIPTION

The `GetTrackUsage` function returns a long integer that contains flags indicating the track's usage. The following flags are defined (unused flags are set to 0):

`trackUsageInMovie`

The track is used in the movie. If this flag is set to 1, the track is used in the movie.

`trackUsageInPreview`

The track is used in the movie preview. If this flag is set to 1, the track is used in the preview.

`trackUsageInPoster`

The track is used in the movie poster. If this flag is set to 1, the track is used in the poster.

ERROR CODES

`invalidTrack` -2009 This track is corrupted or invalid

ShowMoviePoster

You can use the `ShowMoviePoster` function to display a movie's poster. The movie poster uses the movie's matrix and display clipping characteristics.

```
pascal void ShowMoviePoster (Movie theMovie);
```

theMovie Specifies the movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

DESCRIPTION

The Movie Toolbox draws the movie poster once, in the movie’s graphics world. This function works on active and inactive movies.

ERROR CODES

invalidMovie -2010 This movie is corrupted or invalid

SEE ALSO

You can set the poster’s starting time with the `SetMoviePosterTime` function (described on page 2-118). You can set the position and size of the poster by calling the `SetPosterBox` function (described in the next section).

SetPosterBox

You can use the `SetPosterBox` function to set a poster’s boundary rectangle. You define the poster’s image by specifying a time in the movie (use the `SetMoviePosterTime` function, described on page 2-118). You specify the size and position of the poster image with this function.

```
pascal void SetPosterBox (Movie theMovie, const Rect *boxRect);
```

- `theMovie` Specifies the movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).
- `boxRect` Contains a pointer to a rectangle. The Movie Toolbox sets the poster’s boundary rectangle to the coordinates specified in the structure referred to by this parameter.

DESCRIPTION

If you do not specify a boundary rectangle for the poster, the Movie Toolbox uses the movie’s matrix when it displays the poster.

ERROR CODES

invalidMovie -2010 This movie is corrupted or invalid
invalidRect -2036 Specified rectangle has invalid coordinates

SEE ALSO

Your application can retrieve a poster’s boundary rectangle by calling the `GetPosterBox` function, which is described in the next section.

GetPosterBox

The `GetPosterBox` function allows you to obtain a poster's boundary rectangle.

```
pascal void GetPosterBox (Movie theMovie, Rect *boxRect);
```

theMovie Specifies the movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

boxRect Contains a pointer to a rectangle. The Movie Toolbox returns the poster's boundary rectangle into the structure referred to by this parameter.

DESCRIPTION

When you call `GetPosterBox` without having called `SetPosterBox`, the current movie matrix is applied to the poster tracks to determine the poster box.

ERROR CODES

`invalidMovie` -2010 This movie is corrupted or invalid

SEE ALSO

You set the poster's boundary rectangle by calling the `SetPosterBox` function, which is described in the previous section.

SetMoviePosterTime

The `SetMoviePosterTime` function sets the poster time for the movie. Since a movie poster is a still frame, it is defined by a point in time within the movie. The poster's time is expressed in the movie's time coordinate system. Your application can retrieve a poster's time by calling the `GetMoviePosterTime` function, which is described in the next section.

```
pascal void SetMoviePosterTime (Movie theMovie,
                                TimeValue posterTime);
```

theMovie Specifies the movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

`posterTime` Contains the starting time for the movie frame that contains the poster image.

ERROR CODES

<code>invalidMovie</code>	-2010	This movie is corrupted or invalid
<code>invalidTime</code>	-2015	This time value is invalid

SEE ALSO

Your application can set the poster’s boundary rectangle by calling the `SetPosterBox` function, which is described on page 2-117.

GetMoviePosterTime

The `GetMoviePosterTime` function returns the poster’s time in the movie. Since a movie poster has no duration, a poster is defined by a point in time within the movie. The time value returned is in the time coordinate system of the movie.

`pascal TimeValue GetMoviePosterTime (Movie theMovie);`

`theMovie` Specifies the movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

DESCRIPTION

The `GetMoviePosterTime` function returns a time value. This time value contains the starting time for the movie frame that contains the movie poster image.

ERROR CODES

<code>invalidMovie</code>	-2010	This movie is corrupted or invalid
---------------------------	-------	------------------------------------

SEE ALSO

Your application can set a poster’s time by calling the `SetMoviePosterTime` function, which is described in the previous section.

PlayMoviePreview

The `PlayMoviePreview` function plays a movie's preview.

```
pascal void PlayMoviePreview (Movie theMovie,
                              MoviePreviewCallOutProc callOutProc,
                              long refcon);
```

theMovie Specifies the movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

callOutProc Contains a pointer to a movie callout function in your application. The Movie Toolbox calls this function repeatedly while the movie preview is playing. You can use this function to stop the preview. If you do not want to assign a function, set this parameter to `nil`.

Your function should have the following form:

```
pascal Boolean MyCallOutProc (long refcon);
```

The `refCon` parameter contains the reference constant you specified when you called the `PlayMoviePreview` function.

Your function returns a Boolean value. The Movie Toolbox examines this value before continuing. If your function sets this value to `false`, the Movie Toolbox stops the preview and returns to your application. For details, see “Movie Callout Functions” on page 2-359.

Note that if you call the `GetMovieActiveSegment` function (described on page 2-137) from within your movie callout function, the Movie Toolbox will have changed the active movie segment to be the preview segment of the movie. The Movie Toolbox restores the active segment when the preview is done playing.

refcon Contains a reference constant for your function. The Movie Toolbox passes this value to your function.

DESCRIPTION

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. The Movie Toolbox plays the preview in the movie's graphics world.

ERROR CODES

`invalidMovie` `-2010` This movie is corrupted or invalid

SEE ALSO

Use the `SetMoviePreviewTime` function, described on page 2-122, to define the starting time and duration of the movie preview.

SetMoviePreviewMode

The `SetMoviePreviewMode` function allows your application to place a movie into and out of preview mode. When a movie is in preview mode, only those tracks identified as preview tracks are serviced. You specify how a track is used by calling the `SetTrackUsage` function, which is described on page 2-115.

```
pascal void SetMoviePreviewMode (Movie theMovie,
                                Boolean usePreview);
```

`theMovie` Specifies the movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

`usePreview` Specifies the movie's mode. Set this parameter to `true` to place the movie into preview mode. Set this parameter to `false` to place the movie into normal playback mode.

DESCRIPTION

When you place a movie into preview mode, the Movie Toolbox sets the active movie segment to be the preview segment of the movie. When you take a movie out of preview mode and place it back in normal playback mode, the toolbox sets the active movie segment to be the entire movie. For information about working with active movie segments, see “Enhancing Movie Playback Performance” beginning on page 2-134.

ERROR CODES

`invalidMovie` `-2010` This movie is corrupted or invalid

GetMoviePreviewMode

The `GetMoviePreviewMode` function allows your application to determine whether a movie is in preview mode. If a movie is in preview mode, only the movie's preview can be displayed. Your application can place a movie into and out of preview mode by calling the `SetMoviePreviewMode` function, which is described in the previous section.

```
pascal Boolean GetMoviePreviewMode (Movie theMovie);
```

theMovie Specifies the movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

DESCRIPTION

The `GetMoviePreviewMode` function returns a Boolean value. If the movie is in preview mode, the function sets this return value to `true`. If the movie is in normal playback mode, the function sets this value to `false`.

ERROR CODES

`invalidMovie` -2010 This movie is corrupted or invalid

SetMoviePreviewTime

The `SetMoviePreviewTime` function allows your application to define the starting time and duration of the movie's preview. These time values are in the movie's time coordinate system.

```
pascal void SetMoviePreviewTime (Movie theMovie,
                                TimeValue previewTime,
                                TimeValue previewDuration);
```

theMovie Specifies the movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

previewTime Contains a time value that specifies the preview's starting time.

previewDuration Contains a time value that specifies the preview's duration.

ERROR CODES

invalidMovie	-2010	This movie is corrupted or invalid
invalidDuration	-2014	This duration value is invalid
invalidTime	-2015	This time value is invalid

SEE ALSO

Your application can retrieve the starting time and duration of the preview with the `GetMoviePreviewTime` function, which is described in the next section.

GetMoviePreviewTime

The `GetMoviePreviewTime` function returns the starting time and duration of the movie’s preview. These time values are expressed in the movie’s time coordinate system.

```
pascal void GetMoviePreviewTime (Movie theMovie,
                                TimeValue *previewTime,
                                TimeValue *previewDuration);
```

theMovie Specifies the movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

previewTime Contains a pointer to a time value. The Movie Toolbox places the preview’s starting time into the field referred to by this parameter. If the movie does not have a preview, the Movie Toolbox sets this returned value to 0.

previewDuration Contains a pointer to a time value. The Movie Toolbox places the preview’s duration into the field referred to by this parameter. If the movie does not have a preview, the Movie Toolbox sets this returned value to 0.

ERROR CODES

invalidMovie	-2010	This movie is corrupted or invalid
--------------	-------	------------------------------------

SEE ALSO

Your application sets the starting time and duration of the movie preview with the `SetMoviePreviewTime` function, which is described in the previous section.

Movies and Your Event Loop

In order for your movies to play, your application must grant time to the Movie Toolbox. You do this by calling the `MoviesTask` function from your main event loop. The `MoviesTask` function causes the Movie Toolbox to service all your active movies. You should call this function regularly so that your movie can play smoothly. You can use the `UpdateMovie` function to force your movie to be redrawn after it has been uncovered.

You may want your application to take a particular action when a movie is done playing. The Movie Toolbox provides the `IsMovieDone` function, which allows you to determine whether a movie is done playing. The Movie Toolbox also provides more sophisticated callback mechanisms, which are discussed in “Time Base Functions” beginning on page 2-315.

The Movie Toolbox provides two functions that allow your application to determine whether a specified point lies in either a movie or a track. Use the `PtInMovie` function with movies; use the `PtInTrack` function with tracks.

Your application can retrieve some status information about movies and tracks. Use the `GetMovieStatus` function to retrieve movie status; use the `GetTrackStatus` function to get track status.

MoviesTask

The `MoviesTask` function services active movies.

```
pascal void MoviesTask (Movie theMovie, long maxMilliSecToUse);
```

theMovie Specifies the movie for this operation. If you set this parameter to `nil`, the Movie Toolbox services all of your active movies. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

maxMilliSecToUse Determines the maximum number of milliseconds that `MoviesTask` can work before returning. If this parameter is 0, `MoviesTask` services every active movie exactly once and then returns. If the parameter is nonzero, `MoviesTask` services as many movies as it can in the allotted time before returning.

Once the `MoviesTask` function starts servicing a movie, it cannot stop until it has completely met the requirements of the movie. Consequently, the `MoviesTask` function may execute for a longer time than that specified in `maxMilliSecToUse`. However, the function does not start servicing a new movie if the time specified by `maxMilliSecToUse` has elapsed.

Movie Toolbox

The preferred way to use `MoviesTask` is to set the `maxMilliSecToUse` parameter to 0; however, if you just want to play one movie, you can call `MoviesTask` on that one.

If your rate is 0, `MoviesTask` draws that frame and no other.

DESCRIPTION

When servicing a movie, the Movie Toolbox performs the processing that is appropriate for the movie—displaying frames, playing sound, reading data from disk, or other tasks. The only time the Movie Toolbox actually draws a movie is during the operation of the `MoviesTask` function.

You should call `MoviesTask` as often as possible from your application's main event loop. Note that you should call this function after you have performed your own event processing.

The `MoviesTask` function services only active movies, and only enabled tracks within those active movies. Use the `SetMovieActive` function (described on page 2-145) and the `SetTrackEnabled` function (described on page 2-147) to enable and disable movies and tracks.

SPECIAL CONSIDERATIONS

Note that the `MoviesTask` function services only your movies. Your application must call the Event Manager's `WaitNextEvent` routine (or the Event Manager's `GetNextEvent` routine and the `SystemTask` routine) to give other applications the opportunity to call `MoviesTask` for their movies. For details on `WaitNextEvent`, `GetNextEvent`, and `SystemTask`, see *Inside Macintosh: Macintosh Toolbox Essentials*.

ERROR CODES

<code>invalidMovie</code>	-2010	This movie is corrupted or invalid
---------------------------	-------	------------------------------------

IsMovieDone

Your application may wish to take a particular action when a movie is done playing. The `IsMovieDone` function allows you to determine if a particular movie has completely finished playing.

```
pascal Boolean IsMovieDone (Movie theMovie);
```

<code>theMovie</code>	Specifies the movie for this operation. Your application obtains this movie identifier from such functions as <code>NewMovie</code> , <code>NewMovieFromFile</code> , and <code>NewMovieFromHandle</code> (described on page 2-92, page 2-88, and page 2-90, respectively).
-----------------------	---

Movie Toolbox

DESCRIPTION

The `IsMovieDone` function returns `true` if the specified movie has finished playing; otherwise it returns `false`. A movie with a positive rate (playing forward) is considered done when its movie time reaches the movie end time. Conversely, a movie with a negative rate (playing backward) is considered done when its movie time reaches the movie start time.

If your application has changed the movie's active segment, the status returned by the `IsMovieDone` function is relative to the active segment, rather than to the entire movie. You can use the `SetMovieActiveSegment` function (described on page 2-136) to change a movie's active segment.

ERROR CODES

<code>invalidMovie</code>	-2010	This movie is corrupted or invalid
---------------------------	-------	------------------------------------

UpdateMovie

The `UpdateMovie` function allows your application to ensure that the Movie Toolbox properly displays your movie after it has been uncovered.

Your application should call this function between the Window Manager's `BeginUpdate` and `EndUpdate` functions. (For details, see *Inside Macintosh: Macintosh Toolbox Essentials*.) Do not call `MoviesTask` at this time. You will observe better display behavior if you call `MoviesTask` at the end of your update processing.

```
pascal OSErr UpdateMovie (Movie theMovie);
```

<code>theMovie</code>	Specifies the movie for this operation. Your application obtains this movie identifier from such functions as <code>NewMovie</code> , <code>NewMovieFromFile</code> , and <code>NewMovieFromHandle</code> (described on page 2-92, page 2-88, and page 2-90, respectively).
-----------------------	---

DESCRIPTION

The `UpdateMovie` function does not actually update the movie's graphics world. Rather, the function invalidates the movie's display state so that the Movie Toolbox redraws the movie the next time you call the `MoviesTask` function. If you need to force a movie to be redrawn outside of a Window Manager update sequence, your application can call `UpdateMovie` and then call the `MoviesTask` function (described on page 2-124) to service the movie.

The Movie Toolbox determines the portion of the screen to update by examining the graphics port's visible region.

ERROR CODES

`invalidMovie` -2010 This movie is corrupted or invalid

SEE ALSO

For sample code that uses the `UpdateMovie` function in a Window Manager update sequence, see Listing 2-13 on page 2-63.

PtInMovie

The `PtInMovie` function allows your application to determine whether a specified point lies in the region defined by a movie's final display boundary region after it has been clipped by the movie's display clipping region. This function is accurate at the current movie time.

```
pascal Boolean PtInMovie (Movie theMovie, Point pt);
```

<code>theMovie</code>	Specifies the movie for this operation. Your application obtains this movie identifier from such functions as <code>NewMovie</code> , <code>NewMovieFromFile</code> , and <code>NewMovieFromHandle</code> (described on page 2-92, page 2-88, and page 2-90, respectively).
<code>pt</code>	Specifies the point to be checked. This point must be expressed in the movie's local display coordinate system.

DESCRIPTION

The `PtInMovie` function returns a Boolean value. The function sets this value to `true` if the point lies in the movie's display space.

SPECIAL CONSIDERATIONS

The region that `PtInMovie` checks for is different from the movie box.

ERROR CODES

`invalidMovie` -2010 This movie is corrupted or invalid

SEE ALSO

To find out if a point lies in the region defined by a track's display boundary region after it has been clipped by a movie's final display clipping region, you use the `PtInTrack` function. See the next section for details.

PtInTrack

The `PtInTrack` function allows your application to determine whether a specified point lies in the region defined by a track's display boundary region after it has been clipped by the movie's final display clipping region. This function is accurate at the current movie time.

```
pascal Boolean PtInTrack (Track theTrack, Point pt);
```

`theTrack` Specifies the track for this operation. Your application obtains this track identifier from such Movie Toolbox functions as `NewMovieTrack` and `GetMovieTrack` (described on page 2-151 and page 2-204, respectively).

`pt` Specifies the point to be checked. This point must be expressed in the local display coordinate system of the movie that contains the track.

DESCRIPTION

The `PtInTrack` function returns a Boolean value. The function sets this value to `true` if the point lies in the track's display space.

SPECIAL CONSIDERATIONS

The region that `PtInTrack` checks for is different from the movie box.

ERROR CODES

<code>invalidMovie</code>	-2010	This movie is corrupted or invalid
---------------------------	-------	------------------------------------

SEE ALSO

To find out if a point lies within the region defined by a movie's final display boundary region after it has been clipped by the movie's display clipping region, you can use the `PtInMovie` function, which is described in the previous section.

GetMovieStatus

The `GetMovieStatus` function searches for errors in all the enabled tracks of the movie. This function returns information about errors that are encountered during the processing associated with the `MoviesTask` function (described on page 2-124). These errors typically reflect playback problems, such as low-memory conditions.

```
pascal ComponentResult GetMovieStatus (Movie theMovie,
                                         Track *firstProblemTrack);
```

Movie Toolbox

`theMovie` Specifies the movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

`firstProblemTrack` Contains a pointer to a track identifier. The Movie Toolbox places the identifier for the first track that is found to contain an error into the field referred to by this parameter. If you do not want to receive the track identifier, set this parameter to `nil`.

DESCRIPTION

The `GetMovieStatus` function returns the error from the first problem track. If the component does not find any errors, the result is set to `noErr`.

ERROR CODES

Any Movie Toolbox result code (see “Summary of the Movie Toolbox” at the end of this chapter)

GetTrackStatus

The `GetTrackStatus` function returns the value of the last error the media encountered while playing a specified track. This function returns information about errors that are encountered during the processing associated with the `MoviesTask` function (described on page 2-124). These errors typically reflect playback problems, such as low-memory conditions.

The media clears this error code when it detects that the error has been corrected.

```
pascal ComponentResult GetTrackStatus (Track theTrack);
```

`theTrack` Specifies the track for this operation. Your application obtains this track identifier from the `GetMovieStatus` function, described in the previous section.

DESCRIPTION

The `GetTrackStatus` function returns the last error encountered for the specified track. If the component does not find any errors, the result is set to `noErr`.

ERROR CODES

Any Movie Toolbox result code (see “Summary of the Movie Toolbox” at the end of this chapter)

Preferred Movie Settings

Every movie has default, or preferred, settings for playback rate and volume. These settings are stored with the movie in its movie file. The Movie Toolbox provides functions that allow your application to manipulate these default settings.

You can use the `GetMoviePreferredRate` and `SetMoviePreferredRate` functions to work with a movie's default playback rate. You can use the `GetMoviePreferredVolume` and `SetMoviePreferredVolume` functions to work with the default sound volume of a movie.

You can use the `SetMovieRate` function to change a movie's playback rate—see “Working with Movie Time” beginning on page 2-184 for a complete description of this function. The Movie Toolbox also provides a number of functions that allow you to change other settings when you play a movie. These functions are discussed in “Functions That Modify Movie Properties” beginning on page 2-157.

SetMoviePreferredRate

The `SetMoviePreferredRate` function allows your application to specify a movie's default playback rate.

```
pascal void SetMoviePreferredRate (Movie theMovie, Fixed rate);
```

theMovie Specifies the movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

rate Specifies the new movie rate as a 32-bit, fixed-point number. Positive integers indicate forward rates and negative integers indicate reverse rates.

DESCRIPTION

The default playback rate is the rate that the `StartMovie` function (described on page 2-111) uses when it starts playing a movie. The default preferred rate of a movie is set to 1.0 (the `kFix1` constant) when the movie is created.

SPECIAL CONSIDERATIONS

Do not set the preferred rate to 0.

ERROR CODES

`invalidMovie` -2010 This movie is corrupted or invalid

SEE ALSO

Your application can obtain the preferred playback rate by calling the `GetMoviePreferredRate` function, which is described in the next section.

You can set the current playback rate of a movie by calling the `SetMovieRate` function, which is described on page 2-187.

GetMoviePreferredRate

The `GetMoviePreferredRate` function returns a movie's default playback rate. This is the rate that the `StartMovie` function uses when it starts playing a movie.

```
pascal Fixed GetMoviePreferredRate (Movie theMovie);
```

`theMovie` Specifies the movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

DESCRIPTION

The `GetMoviePreferredRate` function returns the default movie rate as a 32-bit, fixed-point number. Positive integers indicate forward rates and negative integers indicate reverse rates.

ERROR CODES

`invalidMovie` -2010 This movie is corrupted or invalid

SEE ALSO

Your application can change the preferred playback rate by calling the `SetMoviePreferredRate` function, which is described in the previous section. You can change the current playback rate of a movie by calling the `SetMovieRate` function, which is described on page 2-187.

SetMoviePreferredVolume

The `SetMoviePreferredVolume` function allows your application to set a movie's preferred volume setting.

```
pascal void SetMoviePreferredVolume (Movie theMovie,
                                     short volume);
```

theMovie Specifies the movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

volume Specifies the preferred volume setting of the movie. The volume parameter must contain a 16-bit, fixed-point number that contains the movie's default volume. The high-order 8 bits contain the integer part of the value; the low-order 8 bits contain the fractional part. Volume values range from -1.0 to 1.0. Negative values play no sound but preserve the absolute value of the volume setting. You may find the following constants useful:

`kFullVolume`

Sets the movie to full volume (constant value is 1.0).

`kNoVolume`

Sets the movie to no volume (constant value is 0.0).

DESCRIPTION

Your application can obtain the preferred volume setting by calling the `GetMoviePreferredVolume` function, which is described in the next section. You can change a movie's current volume by calling the `SetMovieVolume` function, which is described on page 2-182.

A movie's tracks may have their own volume settings. Use the `SetTrackVolume` function, described on page 2-183, to set the volume of an individual track. A track's volume is scaled by the movie's volume to produce the track's final volume. Furthermore, the movie's volume is scaled by the sound volume that is returned by the Operating System's `GetSoundVol` routine (described in *Inside Macintosh: More Macintosh Toolbox*). Thus, the user can control the overall volume from the Sound control panel.

ERROR CODES

<code>invalidMovie</code>	-2010	This movie is corrupted or invalid
---------------------------	-------	------------------------------------

SEE ALSO

When a movie is loaded, the current setting is set to preferred volume. The `StartMovie` function (described on page 2-111) uses this volume setting when it starts playing a movie.

GetMoviePreferredVolume

The `GetMoviePreferredVolume` function returns a movie's preferred volume setting.

```
pascal short GetMoviePreferredVolume (Movie theMovie);
```

theMovie Specifies the movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

DESCRIPTION

The `GetMoviePreferredVolume` function returns a 16-bit, fixed-point number that contains the movie's default volume. The high-order 8 bits contain the integer part of the value; the low-order 8 bits contain the fractional part. Volume values range from 0.0 to 1.0.

You can change a movie's current volume by calling the `SetMovieVolume` function, which is described on page 2-182.

A movie's tracks have their own volume settings. Use the `SetTrackVolume` function, described on page 2-183, to set the volume of an individual track. A track's volume is scaled by the movie's volume to produce the track's final volume. Furthermore, the movie's volume is scaled by the sound volume that is returned by the Operating System's `GetSoundVol` routine (described in *Inside Macintosh: More Macintosh Toolbox*). Thus, the user can control the overall volume from the Sound control panel.

ERROR CODES

`invalidMovie` -2010 This movie is corrupted or invalid

SEE ALSO

When a movie is loaded, the current setting is set to preferred volume. The `StartMovie` function (described on page 2-111) uses this volume setting when it starts playing a movie.

Enhancing Movie Playback Performance

There are circumstances in which an application needs to optimize the performance of a movie or a portion of a movie. The Movie Toolbox provides several functions to help in this process.

The first step you can take to enhance movie playback performance is to allow the Movie Toolbox to **preroll** the movie. When the toolbox prerolls a movie, it informs the media handlers that the movie is about to play. The media handlers can then load the appropriate movie data. In this manner, the movie can play smoothly from the start. Use the `PrerollMovie` function to preroll a movie.

The next performance enhancement technique is to load portions of a movie, track, or media into memory, thus reducing or eliminating disk access during playback. Loading the movie into RAM provides most noticeable performance improvements when there is a lot of random access involved in the playback process and the entire movie fits into available memory. Use the `LoadMovieIntoRam`, `LoadTrackIntoRam`, and `LoadMediaIntoRam` functions to copy all or part of a movie into memory.

Note

The `LoadMovieIntoRam`, `LoadTrackIntoRam`, and `LoadMediaIntoRam` functions load tracks into memory in a time-slice order so that, if a function fails because it is out of memory, all tracks are left loaded to about the same point in time. ♦

You can influence the temporal accuracy, and therefore the speed, with which the Movie Toolbox tries to display a movie by calling either the `SetMoviePlayHints` or `SetMediaPlayHints` function.

For each movie currently in use, the Movie Toolbox maintains an active movie segment. The active movie segment is the part of the movie that your application is interested in playing. By default, the active movie segment is set to be the entire movie. You may wish to change this to be some segment of the movie—for example, if you wish to play a user's selection repeatedly. By setting the active movie segment you guarantee that the Movie Toolbox uses no samples from outside of that range while playing the movie. Use the `SetMovieActiveSegment` and `GetMovieActiveSegment` functions to work with the active segment.

Some movies contain very few key frames and a great number of frame differences. These movies play back very well because they have a lower data rate. Unfortunately, this makes random access operations, such as scrubbing, on a movie difficult. In such movies, random access is difficult.

To improve random access performance of movies with few key frames and many frame differences, shadow sync samples may be added. **Shadow sync samples** are self-contained samples that are alternates for already existing frame difference samples.

During certain random access operations, a shadow sync sample is used instead of a normal key frame, which may be very far away from the desired frame.

The Movie Toolbox provides two functions to let you create just such an association between a frame difference sample and a sync sample. `SetMediaShadowSync` establishes a shadow sync sample for a media. You can use `GetMediaShadowSync` to find out if a particular frame difference sample has a shadow sync sample.

PrerollMovie

The `PrerollMovie` function allows your application to prepare a portion of a movie for playback.

```
pascal OSErr PrerollMovie (Movie theMovie, TimeValue time,
                          Fixed Rate);
```

- `theMovie` Specifies the movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).
- `time` Contains the starting time of the movie segment to play.
- `Rate` Specifies the rate at which you anticipate playing the movie. You specify the movie rate as a 32-bit, fixed-point number. Positive integers indicate forward rates and negative integers indicate reverse rates.

DESCRIPTION

When your application calls the `PrerollMovie` function, the Movie Toolbox tells the appropriate media handlers to prepare to play the movie. The media handlers may then load the movie data and perform any other necessary preparations to play the movie, such as allocating sound channels and starting up image-decompression sequences. In this manner, you can eliminate playback stutter when the movie starts playing.

ERROR CODES

<code>invalidMovie</code>	-2010	This movie is corrupted or invalid
<code>invalidTime</code>	-2015	This time value is invalid

SetMovieActiveSegment

You can use the `SetMovieActiveSegment` function to define a movie's active segment. Your application defines the active segment by specifying the starting time and duration of the segment. These values must be expressed in the movie's time coordinate system. By default, the entire movie is active.

```
pascal void SetMovieActiveSegment (Movie theMovie,
                                   TimeValue startTime,
                                   TimeValue duration);
```

<code>theMovie</code>	Specifies the movie for this operation. Your application obtains this movie identifier from such functions as <code>NewMovie</code> , <code>NewMovieFromFile</code> , and <code>NewMovieFromHandle</code> (described on page 2-92, page 2-88, and page 2-90, respectively).
<code>startTime</code>	Contains a time value specifying the starting point of the active segment. Set this parameter to -1 to make the entire movie active. In this case, the <code>SetMovieActiveSegment</code> function ignores the <code>duration</code> parameter.
<code>duration</code>	Contains a time value that specifies the duration of the active segment. If you are making the entire movie active (by setting the <code>startTime</code> parameter to -1), the Movie Toolbox ignores this parameter.

DESCRIPTION

Your application can retrieve the information that defines a movie's active segment by calling the `GetMovieActiveSegment` function, which is described in the next section.

SPECIAL CONSIDERATIONS

Note that placing a movie into preview mode destroys the movie's active segment. You use the `SetMoviePreviewMode` function, described on page 2-121, to control preview mode.

ERROR CODES

<code>invalidMovie</code>	-2010	This movie is corrupted or invalid
<code>invalidDuration</code>	-2014	This duration value is invalid
<code>invalidTime</code>	-2015	This time value is invalid

GetMovieActiveSegment

Use the `GetMovieActiveSegment` function to determine what portion of a movie is currently active for playing.

```
pascal void GetMovieActiveSegment (Movie theMovie,
                                   TimeValue *startTime,
                                   TimeValue *duration);
```

<code>theMovie</code>	Specifies the movie for this operation. Your application obtains this movie identifier from such functions as <code>NewMovie</code> , <code>NewMovieFromFile</code> , and <code>NewMovieFromHandle</code> (described on page 2-92, page 2-88, and page 2-90, respectively).
<code>startTime</code>	Contains a pointer to a time value. The <code>GetMovieActiveSegment</code> function places the starting time of the active segment into the field referred to by this parameter. If the returned time value is set to -1, the entire movie is active. In this case, the Movie Toolbox does not return any duration information via the <code>duration</code> parameter.
<code>duration</code>	Contains a pointer to a time value. The <code>GetMovieActiveSegment</code> function places the duration of the active movie segment into the field referred to by this parameter. If the entire movie is active (the returned starting time is set to -1), the Movie Toolbox does not return any duration information.

DESCRIPTION

Your application can set the active segment by calling the `SetMovieActiveSegment` function, which is described in the previous section.

ERROR CODES

<code>invalidMovie</code>	-2010	This movie is corrupted or invalid
---------------------------	-------	------------------------------------

SetMoviePlayHints

The `SetMoviePlayHints` function allows your application to provide information to the Movie Toolbox that can influence movie playback. This function accepts a flag in which you specify optimizations that the Movie Toolbox can use during movie playback. These optimizations apply to all of the media structures used by the movie.

```
pascal void SetMoviePlayHints (Movie theMovie, long flags,
                               long flagsMask);
```

Movie Toolbox

<code>theMovie</code>	Specifies the movie for this operation. Your application obtains this movie identifier from such functions as <code>NewMovie</code> , <code>NewMovieFromFile</code> , and <code>NewMovieFromHandle</code> (described on page 2-92, page 2-88, and page 2-90, respectively).
<code>flags</code>	Specifies the optimizations that can be used with this movie. Each bit in the <code>flags</code> parameter corresponds to a specific optimization. The following flag is defined (be sure to set unused flags to 0):
<code>hintsScrubMode</code>	<p>Indicates that the Movie Toolbox can prefer to display key frames when the movie is repositioned. This optimization is used only when a movie's rate is set to 0. If you set this flag to 1, the Movie Toolbox is free to display the nearest key frame when you set the movie's current time; the Movie Toolbox then moves to the appropriate frame as time permits. If you set this flag to 0, the Movie Toolbox displays the frame that corresponds to the new current time, even if that frame is not a key frame.</p> <p>By displaying key frames first, the Movie Toolbox can display data from temporally compressed movies much more quickly in response to changes to the movie's current time. This, in turn, can improve the liveliness of a movie control. For example, if the user is positioning in a stopped movie, the Movie Toolbox can display a key frame that corresponds to the new position without having to build up the image offscreen. In this manner, the user gets quicker feedback from your application.</p>
<code>hintsUseSoundInterp</code>	Turns on sound interpolation—that is, tells the Sound Manager to use sound interpolation when playing back sound. In certain situations, this improves the sound quality to 11 kHz.
<code>hintsAllowInterlace</code>	Tells the Image Compression Manager to use the interlace option for image compressor and decompressor components. For more information, see <i>Inside Macintosh: QuickTime Components</i> .
<code>flagsMask</code>	Indicates which flags in the <code>flags</code> parameter are to be considered in this operation. For each bit in the <code>flags</code> parameter that you want the Movie Toolbox to consider, you must set the corresponding bit in the <code>flagsMask</code> parameter to 1. Set unused flags to 0. This allows you to work with a single optimization without altering the settings of other flags.

ERROR CODES

<code>invalidMovie</code>	-2010	This movie is corrupted or invalid
---------------------------	-------	------------------------------------

SetMediaPlayHints

The `SetMediaPlayHints` function allows your application to provide information to the Movie Toolbox that can influence playback of a single media. This function accepts a flag in which you specify optimizations that the Movie Toolbox can use during movie playback. These optimizations apply to only the specified media.

```
pascal void SetMediaPlayHints (Media theMedia, long flags,
                               long flagsMask);
```

theMedia Specifies the media for this operation. Your application obtains this media identifier from such Movie Toolbox functions as `NewTrackMedia` and `GetTrackMedia` (described on page 2-153 and page 2-206, respectively).

flags Specifies the optimizations that can be used with this media. Each bit in the `flags` parameter corresponds to a specific optimization. The following flag is defined (be sure to set unused flags to 0):

`hintsScrubMode`

Indicates that the Movie Toolbox can prefer to display key frames when the movie that uses this media is repositioned. This optimization is used only when a movie's rate is set to 0. If you set this flag to 1, the Movie Toolbox is free to display the nearest key frame when you set the movie's current time; the Movie Toolbox then moves to the appropriate frame as time permits. If you set this flag to 0, the Movie Toolbox displays the frame that corresponds to the new current time, even if that frame is not a key frame.

By displaying key frames first, the Movie Toolbox can display data from temporally compressed movies much more quickly in response to changes to the movie's current time. This, in turn, can improve the liveliness of a movie control. For example, if the user is positioning in a stopped movie, the Movie Toolbox can display a key frame that corresponds to the new position without having to build up the image offscreen. In this manner, the user gets quicker feedback from your application.

`hintsUseSoundInterp`

Turns on sound interpolation—that is, tells the Sound Manager to use sound interpolation when playing back sound. In certain situations, this improves the sound quality to 11 kHz.

`hintsAllowInterlace`

Tells the Image Compression Manager to use the interlace option for image compressor and decompressor components. For more information, see *Inside Macintosh: QuickTime Components*.

Movie Toolbox

flagsMask Indicates which flags in the `flags` parameter are to be considered in this operation. For each bit in the `flags` parameter that you want the Movie Toolbox to consider, you must set the corresponding bit in the `flagsMask` parameter to 1. Set unused flags to 0. This allows you to work with a single optimization without altering the settings of other flags.

ERROR CODES

`invalidMedia` -2008 This media is corrupted or invalid

SEE ALSO

To set optimizations for all of a movie's media structures, use the `SetMoviePlayHints` function, which is described in the previous section.

LoadMovieIntoRam

The `LoadMovieIntoRam` function loads a movie's data into memory. If the movie does not fit, the function returns an error.

```
pascal OSErr LoadMovieIntoRam (Movie theMovie, TimeValue time,
                                TimeValue duration,
                                long flags);
```

theMovie Specifies the movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

time Allows you to specify a portion of the movie to load. The `time` parameter contains the starting time of the movie segment to load. The `duration` parameter specifies the length of the segment to load.

duration Allows you to specify a portion of the movie to load. The `time` parameter contains the starting time of the movie segment to load. The `duration` parameter specifies the length of the segment to load. You can use the `GetMovieDuration` function (described on page 2-185) to determine the length of the entire movie. Note that the Movie Toolbox may load more data than you specify due to the way the data is loaded.

flags Gives you explicit control over what is loaded into memory and how long to keep it around. The following constants are provided. You can set these flags in any combination that makes sense for you.

`keepInRam`

Renders all data loaded with this flag set as nonpurgeable. Nonpurgeable data is not released from memory until you request it explicitly. This practice can fill up your heap very quickly. Exercise caution.

- `unkeepInRam`
Renders all indicated data purgeable. The data is not necessarily released from memory immediately, however. Information about whether a chunk can be purged is maintained internally by a single bit. This means there is no counter. Therefore, if you care very much about the data, you have to work very hard and use the edit list meticulously.
- `flushFromRam`
Purges all indicated data from memory, unless it is currently in use by a media handler (for example, if it is still drawing frames from the requested times). This flag makes the memory available for purging, and then performs the purge. You may want to use this option if you are particularly low on memory.
- `loadForwardTrackEdits`
In some cases, an edited movie plays back much more smoothly if the data around edits is already in RAM. By setting either this flag or the `lookBackwardTrackEdits` flag, you can load only the data around edits. The Movie Toolbox walks through the edits and decides the right amount of data to load for you. If you are going to play the movie forward, set only the `loadForwardTrackEdits` flag. If you are going to play in both directions, or you don't know which direction, set both flags.
- `loadBackwardTrackEdits`
In some cases, an edited movie plays back much more smoothly if the data around edits is already in RAM. By setting either this flag or `lookForwardTrackEdits`, you can load only the data around edits. The Movie Toolbox walks through the edits and decides the right amount of data to load for you. If you are going to play the movie only backward, set the `loadBackwardTrackEdits` flag. If you are going to play in both directions, or you don't know which direction, set both flags.

DESCRIPTION

If `LoadMovieIntoRam` fails because it was out of memory, no data is purged.

ERROR CODES

- | | | |
|----------------------------------|-------|--|
| <code>invalidMovie</code> | -2010 | This movie is corrupted or invalid |
| <code>invalidDuration</code> | -2014 | This duration value is invalid |
| <code>invalidTime</code> | -2015 | This time value is invalid |
| <code>progressProcAborted</code> | -2019 | Your progress function returned an error |
| File Manager errors | | |
| Memory Manager errors | | |

LoadTrackIntoRam

The `LoadTrackIntoRam` function loads a track's data into memory. If the track does not fit, the function returns an error.

```
pascal OSErr LoadTrackIntoRam (Track theTrack, TimeValue time,
                               TimeValue duration, long flags);
```

- | | |
|-----------------------|--|
| <code>theTrack</code> | Specifies the track for this operation. Your application obtains this track identifier from such Movie Toolbox functions as <code>NewMovieTrack</code> and <code>GetMovieTrack</code> (described on page 2-151 and page 2-204, respectively). |
| <code>time</code> | Allows you to specify a portion of the track to load. The time parameter contains the starting time of the track segment to load. The duration parameter specifies the length of the segment to load. You must specify this time value in the movie's time coordinate system. |
| <code>duration</code> | Allows you to specify a portion of the track to load. The time parameter contains the starting time of the track segment to load. The duration parameter specifies the length of the segment to load. You can use the <code>GetTrackDuration</code> function (described on page 2-191) to determine the length of the entire movie. Note that the media handler may load more data than you specify. |
| <code>flags</code> | Gives you explicit control over what is loaded into memory and how long to keep it around. The following constants are provided: |

```
enum
{
    keepInRam = 1<<0,
    unkeepInRam = 1<<1,
    flushFromRam = 1<<2,
    loadForwardTrackEdits = 1<<3,
    loadBackwardTrackEdits = 1<<4
};
```

You can set these flags in any combination that makes sense. For descriptions of the individual flag constants, see the description of the `LoadMovieIntoRam` function on page 2-140.

ERROR CODES

<code>invalidTrack</code>	-2009	This track is corrupted or invalid
<code>invalidDuration</code>	-2014	This duration value is invalid
<code>invalidTime</code>	-2015	This time value is invalid
<code>progressProcAborted</code>	-2019	Your progress function returned an error
File Manager errors		
Memory Manager errors		

LoadMediaIntoRam

The `LoadMediaIntoRam` function loads a media's data into memory.

The exact behavior of `LoadMediaIntoRam` is dependent on the media handler.

```
pascal OSErr LoadMediaIntoRam (Media theMedia, TimeValue time,
                                TimeValue duration, long flags);
```

<code>theMedia</code>	Specifies the media for this operation. Your application obtains this media identifier from such Movie Toolbox functions as <code>NewTrackMedia</code> and <code>GetTrackMedia</code> (described on page 2-153 and page 2-206, respectively).
<code>time</code>	Allows you to specify a portion of the media to load. The time parameter contains the starting time of the media segment to load. The duration parameter specifies the length of the segment to load. This time value must be expressed in the media's time coordinate system.
<code>duration</code>	Allows you to specify a portion of the media to load. The time parameter contains the starting time of the media segment to load. The duration parameter specifies the length of the segment to load. You can use the <code>GetMediaDuration</code> function (described on page 2-194) to determine the length of the entire media. Note that the media handler may load more data than you specify if the media data was added in larger pieces.
<code>flags</code>	Gives you explicit control over what is loaded into memory and how long to keep it around. The following constants are provided:

```
enum
{
    keepInRam = 1<<0,
    unkeepInRam = 1<<1,
    flushFromRam = 1<<2,
};
```

You can set these flags in any combination that makes sense. For descriptions of the individual flag constants, see the description of the `LoadMovieIntoRam` function on page 2-140.

DESCRIPTION

If the `LoadMediaIntoRam` function fails because it is out of memory, no data is purged.

ERROR CODES

<code>invalidMedia</code>	-2008	This media is corrupted or invalid
<code>invalidDuration</code>	-2014	This duration value is invalid
<code>invalidTime</code>	-2015	This time value is invalid
<code>progressProcAborted</code>	-2019	Your progress function returned an error
File Manager errors		
Memory Manager errors		

SetMediaShadowSync

The `SetMediaShadowSync` function creates an association between the indicated frame difference sample and a specified self-contained sample in a given media. This association makes the self-contained sample a shadow sync sample for the frame difference sample.

```
pascal OSErr SetMediaShadowSync (Media theMedia,
                                long frameDiffSampleNum,
                                long syncSampleNum);
```

`theMedia` The media in which the shadow sync is to be created.

`frameDiffSampleNum`
 Specifies a frame difference sample. The sample number is obtained from the `MediaTimeToSampleNum` function.

`syncSampleNum`
 Specifies a shadow sync sample. The sample number is obtained from the `MediaTimeToSampleNum` function.

DESCRIPTION

Note that the association established is between sample numbers—not sample times.

SPECIAL CONSIDERATIONS

Shadow sync samples should not be part of a track. You should not call `InsertMediaIntoTrack` on these media samples. Typically, you add shadow sync samples after a media is completely created. Shadow sync samples are not maintained when editing or flattening movies.

ERROR CODES

Memory Manager errors

GetMediaShadowSync

The `GetMediaShadowSync` function returns the sample number of the shadow sync associated with a given frame difference sample number.

```
pascal OSErr GetMediaShadowSync (Media theMedia,
                                long frameDiffSampleNum,
                                long *syncSampleNum);
```

Movie Toolbox

<code>theMedia</code>	Indicates the media in which the shadow sync sample has been established and the shadow sync number is to be obtained.
<code>frameDiffSampleNum</code>	Specifies the frame difference sample number associated with the desired shadow sync sample number.
<code>syncSampleNum</code>	Contains a pointer to the sample number of the shadow sync. If the <code>frameDiffSample</code> parameter does not have a shadow sync, 0 is returned in the <code>syncSampleNum</code> parameter.

ERROR CODES

Memory Manager errors

Disabling Movies and Tracks

The Movie Toolbox services only movies and tracks that are active. This section describes functions that allow your application to enable and disable tracks and movies.

You can use the `SetMovieActive` function to activate and deactivate a movie. Use the `GetMovieActive` function to determine whether a movie is active.

Similarly, your application can use the `SetTrackEnabled` function to enable and disable a track. Use the `GetTrackEnabled` function to determine whether a track is enabled. The Movie Toolbox also allows you to assign alternate tracks based on language or quality criteria. Functions that work with alternate tracks are discussed in “Working With Alternate Tracks” beginning on page 2-207.

SetMovieActive

The `SetMovieActive` function allows your application to activate and deactivate a movie.

```
pascal void SetMovieActive (Movie theMovie, Boolean active);
```

<code>theMovie</code>	Specifies the movie for this operation. Your application obtains this movie identifier from such functions as <code>NewMovie</code> , <code>NewMovieFromFile</code> , and <code>NewMovieFromHandle</code> (described on page 2-92, page 2-88, and page 2-90, respectively).
<code>active</code>	Activates or deactivates the movie. Set this parameter to <code>true</code> to activate the movie; set this parameter to <code>false</code> to deactivate the movie.

SPECIAL CONSIDERATIONS

The Movie Toolbox services only active movies. When you deactivate a movie, the Movie Toolbox may release system resources required by the movie, such as sound hardware, open files, and allocated memory. Unless you set the `newMovieActive` flag when creating a movie, you should call `SetMovieActive` before playing a movie.

ERROR CODES

<code>invalidMovie</code>	-2010	This movie is corrupted or invalid
---------------------------	-------	------------------------------------

SEE ALSO

You can determine whether a movie is active by calling the `GetMovieActive` function, which is described in the next section.

GetMovieActive

The `GetMovieActive` function allows your application to determine whether a movie is currently active. The Movie Toolbox services only active movies.

```
pascal Boolean GetMovieActive (Movie theMovie);
```

<code>theMovie</code>	Specifies the movie for this operation. Your application obtains this movie identifier from such functions as <code>NewMovie</code> , <code>NewMovieFromFile</code> , and <code>NewMovieFromHandle</code> (described on page 2-92, page 2-88, and page 2-90, respectively).
-----------------------	---

DESCRIPTION

The `GetMovieActive` function returns a Boolean value. The function sets this value to `true` if the movie is active and `false` if the movie is not active.

ERROR CODES

<code>invalidMovie</code>	-2010	This movie is corrupted or invalid
---------------------------	-------	------------------------------------

SEE ALSO

You can make a movie active by calling the `SetMovieActive` function, which is described in the previous section.

SetTrackEnabled

The `SetTrackEnabled` function allows your application to enable and disable a track. The Movie Toolbox services only enabled tracks.

```
pascal void SetTrackEnabled (Track theTrack, Boolean isEnabled);
```

theTrack Specifies the track for this operation. Your application obtains this track identifier from such Movie Toolbox functions as `NewMovieTrack` and `GetMovieTrack` (described on page 2-151 and page 2-204, respectively).

isEnabled Enables or disables the track. Set this parameter to `true` to enable the track. Set this parameter to `false` to disable the track.

SPECIAL CONSIDERATIONS

When you disable a track, the Movie Toolbox may release system resources that are used by the track, including allocated memory.

ERROR CODES

`invalidTrack` -2009 This track is corrupted or invalid

SEE ALSO

You can determine whether a track is enabled by calling the `GetTrackEnabled` function, which is described in the next section.

GetTrackEnabled

The `GetTrackEnabled` function allows your application to determine whether a track is currently enabled. The Movie Toolbox services only enabled tracks.

```
pascal Boolean GetTrackEnabled (Track theTrack);
```

theTrack Specifies the track for this operation. Your application obtains this track identifier from such Movie Toolbox functions as `NewMovieTrack` and `GetMovieTrack` (described on page 2-151 and page 2-204, respectively).

DESCRIPTION

The `GetTrackEnabled` function returns a Boolean value. The function sets this value to `true` if the track is enabled and `false` if the track is disabled.

ERROR CODES

`invalidTrack` -2009 This track is corrupted or invalid

SEE ALSO

You can enable a track by calling the `SetTrackEnabled` function, which is described in the previous section.

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.

GetMoviePict

The `GetMoviePict` function creates a picture from the specified movie at the specified time. This function uses only those movie tracks that are currently enabled and would therefore be used in playback. Your application may call this function even if the movie is inactive.

```
pascal PicHandle GetMoviePict (Movie theMovie, TimeValue time);
```

theMovie Specifies the movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

time Specifies the movie image for the picture. The `time` parameter contains the time from which the image is taken.

DESCRIPTION

The `GetMoviePict` function returns a handle to the picture. Your application must dispose of this picture handle by calling QuickDraw's `KillPicture` routine. If the function could not create the picture, the returned handle is set to `nil`.

SPECIAL CONSIDERATIONS

You can use the `GetMoviePict` function to create a picture. If the movie contains compressed data, the picture created by this function may also contain compressed data that cannot be displayed without QuickTime.

ERROR CODES

<code>invalidMovie</code>	-2010	This movie is corrupted or invalid
<code>invalidTime</code>	-2015	This time value is invalid

Image Compression Manager errors

Memory Manager errors

SEE ALSO

If you want to create a picture from a movie's preview, put the movie into preview mode by calling the `SetMoviePreviewMode` function (described on page 2-121), and then call the `GetMoviePict` function.

GetMoviePosterPict

The `GetMoviePosterPict` function creates a picture that contains a movie's poster.

```
pascal PicHandle GetMoviePosterPict (Movie theMovie);
```

<code>theMovie</code>	Specifies the movie for this operation. Your application obtains this movie identifier from such functions as <code>NewMovie</code> , <code>NewMovieFromFile</code> , and <code>NewMovieFromHandle</code> (described on page 2-92, page 2-88, and page 2-90, respectively).
-----------------------	---

DESCRIPTION

The `GetMoviePosterPict` function returns a handle to the picture. Your application must dispose of this picture handle by calling QuickDraw's `KillPicture` routine. If the function could not create the picture, the returned handle is set to `nil`.

SPECIAL CONSIDERATIONS

If you have not assigned a poster time for the movie, the Movie Toolbox creates the poster from the movie image that corresponds to a time value of 0.

ERROR CODES

<code>invalidMovie</code>	-2010	This movie is corrupted or invalid
---------------------------	-------	------------------------------------

Image Compression Manager errors

Memory Manager errors

GetTrackPict

The `GetTrackPict` function creates a QuickDraw picture from the specified track at the specified time. This function is similar to the `GetMoviePict` function (described on page 2-148), except that `GetTrackPict` uses only the specified track to create the picture.

```
pascal PicHandle GetTrackPict (Track theTrack, TimeValue time);
```

theTrack Specifies the track for this operation. Your application obtains this track identifier from such Movie Toolbox functions as `NewMovieTrack` and `GetMovieTrack` (described on page 2-151 and page 2-204, respectively).

time Specifies the track image for the picture. The `time` parameter contains the time from which the image is taken.

DESCRIPTION

The `GetTrackPict` function returns a handle to the picture. Your application must dispose of this picture handle by calling QuickDraw's `KillPicture` routine. If the function could not create the picture, the returned handle is set to `nil`.

SPECIAL CONSIDERATIONS

You can specify a disabled track. If the track contains compressed data, the picture created by this function may also contain compressed data that cannot be displayed without QuickTime.

ERROR CODES

<code>invalidTrack</code>	-2009	This track is corrupted or invalid
<code>invalidTime</code>	-2015	This time value is invalid

Image Compression Manager errors
Memory Manager errors

Creating Tracks and Media Structures

The Movie Toolbox provides several functions that allow your application to create new movie tracks and media structures and to dispose of existing tracks and media structures. You use these functions when you are creating a new movie or when you are editing an existing movie.

You can use the `NewMovieTrack` function to create a new track for a specified movie. Conversely, you can use the `DisposeMovieTrack` function to dispose of an existing track.

Your application can create a new media for a track by calling the `NewTrackMedia` function. You can use the `DisposeTrackMedia` function to dispose of an existing media.

NewMovieTrack

You can create movie tracks by calling the `NewMovieTrack` function. Immediately after creating a new track, you should call the `NewTrackMedia` function to create a media for the track—a track without a media is of no use.

Note that when you add a track to a movie, the Movie Toolbox automatically adjusts the display rectangle of the movie. You may want to detect these changes by calling the `GetMovieBox` function (described on page 2-162) so that you can adjust the size of the movie's display window.

```
pascal Track NewMovieTrack (Movie theMovie, Fixed width,
                             Fixed height, short trackVolume);
```

<code>theMovie</code>	Specifies the movie for this operation. Your application obtains this movie identifier from such functions as <code>NewMovie</code> , <code>NewMovieFromFile</code> , and <code>NewMovieFromHandle</code> (described on page 2-92, page 2-88, and page 2-90, respectively).
<code>width</code>	Specifies a fixed number denoting the display width of the track, in pixels. Along with the <code>height</code> parameter, this parameter defines the track's display rectangle.
<code>height</code>	Specifies a fixed number denoting the display height of the track, in pixels. Together, the <code>height</code> and <code>width</code> parameters define the track's display rectangle. The upper-left corner of this rectangle lies at (0,0) in the movie's rectangle. The <code>height</code> and <code>width</code> parameters therefore establish the lower-right corner of the track's display rectangle. If you are creating a track that is not displayed, such as a sound track, set the <code>height</code> and <code>width</code> parameters to 0.
<code>trackVolume</code>	Specifies the volume setting of the track as a 16-bit, fixed-point number. The high-order 8 bits specify the integer portion; the low-order 8 bits specify the fractional part. Volume values range from -1.0 to 1.0. Negative values play no sound but preserve the absolute value of the volume setting. Set this parameter to <code>kFullVolume</code> to play the track at its full, natural volume. Set this parameter to <code>kNoVolume</code> to set the volume to 0.
<code>kFullVolume</code>	Sets the track to full volume (constant value is 1.0).
<code>kNoVolume</code>	Sets the track to no volume (constant value is 0.0).

Movie Toolbox

DESCRIPTION

The `NewMovieTrack` function returns a track identifier. If the function cannot create the track, it sets the returned identifier to `nil`.

ERROR CODES

<code>invalidMovie</code>	-2010	This movie is corrupted or invalid
Memory Manager errors		

DisposeMovieTrack

The `DisposeMovieTrack` function removes a track from a movie.

```
pascal void DisposeMovieTrack (Track theTrack);
```

<code>theTrack</code>	Specifies the track for this operation. Your application obtains this track identifier from such Movie Toolbox functions as <code>NewMovieTrack</code> and <code>GetMovieTrack</code> (described on page 2-151 and page 2-204, respectively).
-----------------------	---

DESCRIPTION

When you remove a track from a movie, the Movie Toolbox also removes the corresponding media from the movie.

SPECIAL CONSIDERATIONS

Your application should not call this function as part of the process of disposing of a movie. When you dispose of a movie by calling the `DisposeMovie` function (described on page 2-96), the Movie Toolbox disposes of all the movie's tracks and their associated media structures.

ERROR CODES

<code>invalidTrack</code>	-2009	This track is corrupted or invalid
<code>trackNotInMovie</code>	-2030	This track is not in this movie

NewTrackMedia

After you have created a new track, you can create a media for the track by calling the `NewTrackMedia` function. The media refers to the actual data samples used by the track.

```
pascal Media NewTrackMedia (Track theTrack, OSType mediaType,
                             TimeScale timeScale, Handle dataRef,
                             OSType dataRefType);
```

theTrack Specifies the track for this operation. Your application obtains this track identifier from such Movie Toolbox functions as `NewMovieTrack` (described on page 2-151).

mediaType Specifies the type of media to create. The Movie Toolbox uses this value to find the correct media handler for the new media. If the toolbox cannot locate an appropriate media handler, it returns an error. The following types are available:

<code>VideoMediaType</code>	Video media
<code>SoundMediaType</code>	Sound media
<code>TextMediaType</code>	Text media

timeScale Defines the media's time coordinate system.

dataRef Specifies the data reference. This parameter contains a handle to the information that identifies the file that contains this media's data. The type of information stored in that handle depends upon the value of the `dataRefType` parameter.

If you are creating a new media that refers to existing media data, you can use the `GetMediaDataRef` function (described on page 2-217) to obtain information about the existing data reference. You can then supply information about that reference to this function.

Set this parameter to `nil` to use the file that is associated with the movie or if the movie does not have a movie file. For example, if you have created the movie using the `CreateMovieFile` function (described on page 2-96) or the `NewMovieFromFile` function (described on page 2-88), the Movie Toolbox assumes that the movie's data resides in the file specified at that time. If you have created the movie using the `NewMovieFromScrap` or `NewMovie` functions (described on page 2-245 and page 2-92, respectively), the movie does not have a movie file.

dataRefType

Specifies the type of data reference. If the data reference is an alias, you must set this parameter to `rAliasType ('alis')`, indicating that the reference is an alias. See *Inside Macintosh: Files* for more information about aliases and the Alias Manager.

If you are creating a new media that refers to existing media data, you can use the `GetMediaDataRef` function (described on page 2-217) to obtain information about the existing data reference. You can then supply information about that reference to this function.

Movie Toolbox

Set this parameter to `nil` to use the file that is associated with the movie or if the movie does not have a movie file. For example, if you have created the movie using the `CreateMovieFile` function (described on page 2-96) or the `NewMovieFromFile` function (described on page 2-88), the Movie Toolbox assumes that the movie's data resides in the file specified at that time. If you have created the movie using the `NewMovieFromScrap` or `NewMovie` functions (described on page 2-245 and page 2-92, respectively), the movie does not have a movie file.

DESCRIPTION

The `NewTrackMedia` function returns a media identifier. If the function cannot create the new media, it sets this returned value to `nil`.

ERROR CODES

<code>cantFindHandler</code>	-2003	Cannot locate a handler
<code>cantOpenHandler</code>	-2004	Cannot open a handler
<code>noMediaHandler</code>	-2006	Media has no media handler
<code>invalidTrack</code>	-2009	This track is corrupted or invalid
<code>invalidTime</code>	-2015	This time value is invalid

Memory Manager errors

DisposeTrackMedia

The `DisposeTrackMedia` function removes a media from a track. This function does not remove the track from its movie.

```
pascal void DisposeTrackMedia (Media theMedia);
```

theMedia Specifies the media for this operation. Your application obtains this media identifier from such Movie Toolbox functions as `NewTrackMedia` and `GetTrackMedia` (described on page 2-153 and page 2-206, respectively).

SPECIAL CONSIDERATIONS

Your application should not call the `DisposeTrackMedia` function as part of the process of disposing of a movie. When you dispose of a movie by calling `DisposeMovie`, the Movie Toolbox disposes of all the movie's tracks and their associated media structures.

ERROR CODES

<code>invalidMedia</code>	-2008	This media is corrupted or invalid
---------------------------	-------	------------------------------------

Working With Progress and Cover Functions

The Movie Toolbox allows your application to assign two types of custom functions: progress functions and cover functions. These functions allow you to perform special processing under certain circumstances.

Some Movie Toolbox functions can take a long time to execute. For example, if you call the `FlattenMovie` function and specify a large movie, the Movie Toolbox must read and write all the sample data for the movie. During such operations you may wish to display some kind of progress indicator to the user.

A progress function is an application-defined function that you can use to track the progress of time-consuming activities, and thereby keep the user informed about that progress.

The Movie Toolbox allows your application to perform custom processing whenever one of your movie's tracks covers a screen region or reveals a region that was previously covered. You perform this processing in cover functions.

There are two types of cover functions: those that are called when your movie covers a screen region, and those that are called when your movie uncovers a screen region that was previously covered. Cover functions that are called when your movie covers a screen region are responsible for erasing the region—you may choose to save the hidden region in an offscreen buffer. Cover functions that are called when your movie reveals a hidden screen region must redisplay the hidden region.

Note

The Movie Toolbox does not call your cover function in response to changes to the movie's transformation matrix (for example, changing the matrix by calling the `SetMovieBox` function, which is described on page 2-161, does not cause your cover function to be invoked). ♦

For a complete discussion of progress and cover functions, see "Application-Defined Functions," which begins on page 2-354.

The `SetMovieProgressProc` function helps your application work with progress functions and the `SetMovieCoverProcs` function helps your application work with cover functions.

SetMovieProgressProc

The `SetMovieProgressProc` function allows you to attach a progress function to each movie. The function will be called whenever a long operation is underway. The Movie Toolbox indicates the progress of the operation to your progress function.

Movie Toolbox

The Movie Toolbox ensures that your progress function is called regularly, but not too often. In addition, the toolbox calls your function only during long operations.

```
pascal void SetMovieProgressProc (Movie theMovie,
                                   MovieProgressProcPtr p,
                                   long refCon);
```

theMovie	Specifies the movie for this operation. Your application obtains this movie identifier from such functions as <code>NewMovie</code> , <code>NewMovieFromFile</code> , and <code>NewMovieFromHandle</code> (described on page 2-92, page 2-88, and page 2-90, respectively).
p	Points to your progress function. To remove a movie's progress function, set this parameter to <code>nil</code> . Set this parameter to <code>-1</code> for the Movie Toolbox to provide a default progress function. See "Progress Functions" beginning on page 2-354 for the interface your progress function must support.
refCon	Specifies a reference constant. The Movie Toolbox passes this value to your progress function.

DESCRIPTION

The following Movie Toolbox functions use progress functions:

`ConvertFileToMovieFile` (described on page 2-93), `CutMovieSelection` (described on page 2-247), `CopyMovieSelection` (described on page 2-248), `AddMovieSelection` (described on page 2-250), and `InsertMovieSegment` (described on page 2-257).

ERROR CODES

<code>invalidMovie</code>	<code>-2010</code>	This movie is corrupted or invalid
---------------------------	--------------------	------------------------------------

SetMovieCoverProcs

The `SetMovieCoverProcs` function allows you to set both types of cover functions.

```
pascal void SetMovieCoverProcs (Movie theMovie,
                                   MovieRgnCoverProc uncoverProc,
                                   MovieRgnCoverProc coverProc,
                                   long refcon);
```

theMovie	Specifies the movie for this operation. Your application obtains this movie identifier from such functions as <code>NewMovie</code> , <code>NewMovieFromFile</code> , and <code>NewMovieFromHandle</code> (described on page 2-92, page 2-88, and page 2-90, respectively).
----------	---

Movie Toolbox

uncoverProc

Points to a cover function. This function is called whenever one of your movie's tracks is removed from the screen or resized, revealing a previously hidden screen region. If you want to remove the cover function, set this parameter to `nil`. When the `uncoverProc` parameter is `nil`, `SetMovieCoverProcs` uses the default cover or uncover function. The default cover function does nothing. The default uncover function erases the uncovered area. See "Cover Functions" beginning on page 2-357 for the interface your cover function must support.

coverProc

Points to a cover function. The Movie Toolbox calls this function whenever one of your movies covers a portion of the screen. If you want to remove the cover function, set this parameter to `nil`. See "Cover Functions" beginning on page 2-357 for the interface your cover function must support.

refcon

Specifies a reference constant. The Movie Toolbox passes this value to your cover functions.

ERROR CODES

`invalidMovie` -2010 This movie is corrupted or invalid

Functions That Modify Movie Properties

The Movie Toolbox provides a number of functions that allow applications to edit existing movies or to create the contents of new movies. This section describes those functions. It has been divided into the following topics:

- "Working With Movie Spatial Characteristics" describes a number of functions that allow you to work with the display characteristics of movies
- "Working With Sound Volume" discusses the functions that your application can use to work with the sound volume of a movie or a track
- "Working with Movie Time" discusses several functions that allow your application to change the time characteristics of movies
- "Working With Track Time" describes functions that your application can use to change the time characteristics of individual tracks within a movie
- "Working With Media Time" discusses the functions that your application can use to change the time characteristics of a media
- "Finding Interesting Times" describes the Movie Toolbox functions that allow you to retrieve information about when key events occur in movies, tracks, and media structures
- "Locating a Movie's Tracks and Media Structures" describes the functions that allow your application to find tracks that are associated with a movie
- "Working With Alternate Tracks" discusses the Movie Toolbox functions that allow you to define and use alternate tracks in a movie

Movie Toolbox

- “Working With Data References” describes the Movie Toolbox functions that allow you to work with a movie’s data references
- “Determining Movie Creation and Modification Time” discusses the functions that you can use to determine when a movie was created or last changed
- “Working With Media Samples” describes several functions that allow you to get and set detailed information about sample data in a media
- “Working With Movie User Data” discusses the functions that you can use to get and set the user data that is associated with a movie

Working With Movie Spatial Characteristics

The Movie Toolbox provides a number of functions that allow your application to determine and change the display characteristics of movies and tracks. These functions are discussed in the following sections. Before using any of these functions, you should be familiar with the way in which the Movie Toolbox displays movies. See the discussion of spatial properties in “About Movies” on page 2-14.

You can use the `SetMovieGWorld` and `GetMovieGWorld` functions to work with a movie’s graphics world. See *Inside Macintosh: Imaging* for more information about graphics worlds.

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” beginning on page 2-341.

The following 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.

Movie Toolbox

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.

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. ♦

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.

SetMovieGWorld

The `SetMovieGWorld` function allows your application to establish a movie's display coordinate system by setting the graphics world for displaying a movie.

```
pascal void SetMovieGWorld (Movie theMovie, CGrafPtr port,
                           GDHandle gdh);
```

<code>theMovie</code>	Specifies the movie for this operation. Your application obtains this movie identifier from such functions as <code>NewMovie</code> , <code>NewMovieFromFile</code> , and <code>NewMovieFromHandle</code> (described on page 2-92, page 2-88, and page 2-90, respectively).
<code>port</code>	Points to the movie's graphics port structure or graphics world. Set this parameter to <code>nil</code> to use the current graphics port.
<code>gdh</code>	Contains a handle to the movie's graphics device structure. Set this parameter to <code>nil</code> to use the current device. If the <code>port</code> parameter specifies a graphics world, set this parameter to <code>nil</code> to use that graphics world's graphics device.

DESCRIPTION

The default cover function provided by the Movie Toolbox uses the background color and pattern from the movie's graphics world during erase operations.

Movie Toolbox

SPECIAL CONSIDERATIONS

The Movie Toolbox automatically sets the graphics world when you create a new movie. Be sure that your application's graphics port is valid or that you specify a valid graphics port with the `port` parameter. If you pass `nil` for the `port` parameter, make sure the current graphics world is valid.

When you use `SetMovieGWorld`, the Movie Toolbox remembers the current background color and background pattern. These are used for erasing in the default movie uncover function.

ERROR CODES

`invalidMovie` -2010 This movie is corrupted or invalid

SEE ALSO

You can retrieve a movie's graphics world by calling the `GetMovieGWorld` function, which is described in the next section.

GetMovieGWorld

Your application can determine a movie's graphics world by calling the `GetMovieGWorld` function.

```
pascal void GetMovieGWorld (Movie theMovie, CGrafPtr *port,
                             GDHandle *gdh);
```

<code>theMovie</code>	Specifies the movie for this operation. Your application obtains this movie identifier from such functions as <code>NewMovie</code> , <code>NewMovieFromFile</code> , and <code>NewMovieFromHandle</code> (described on page 2-92, page 2-88, and page 2-90, respectively).
<code>port</code>	Contains a pointer to a field that is to receive a pointer to a graphics port structure. The Movie Toolbox returns a pointer to the movie's graphics port structure. Set this parameter to <code>nil</code> if you do not want this information.
<code>gdh</code>	Contains a pointer to a field that is to receive a handle to a graphics device structure. The Movie Toolbox returns a handle to the movie's graphics device structure. Set this parameter to <code>nil</code> if you do not want this information.

ERROR CODES

`invalidMovie` -2010 This movie is corrupted or invalid

SEE ALSO

You can set a movie's graphics world by calling the `SetMovieGWorld` function, which is described in the previous section.

SetMovieBox

The `SetMovieBox` function sets a movie's boundary rectangle, or movie box, which is a rectangle that encompasses the spatial representation of all of the movie's enabled tracks. The movie box is in the display coordinate system.

```
pascal void SetMovieBox (Movie theMovie, const Rect *boxRect);
```

`theMovie` Specifies the movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

`boxRect` Contains a pointer to a rectangle that contains the coordinates of the new boundary rectangle.

DESCRIPTION

The Movie Toolbox changes the rectangle by modifying the translation and scale values of the movie's matrix to accommodate the new boundary rectangle.

The movie box might not have its upper-left corner set at (0,0) in its display window when the movie is first loaded. Consequently, your application may need to adjust the position of the movie box so that it appears in the appropriate location within your application's document window. If you don't reset the movie position, the movie might not be visible when it starts playing.

The following sample code demonstrates how to move the boundary rectangle.

```
GetMovieBox (movie, &movieBox);
OffsetRect (&movieBox, -movieBox.left, -movieBox.top);
SetMovieBox (movie, &movieBox);
```

SPECIAL CONSIDERATIONS

The `SetMovieBox` function does not call your cover functions.

Movie Toolbox

ERROR CODES

`invalidMovie` -2010 This movie is corrupted or invalid
 Memory Manager errors

SEE ALSO

You can modify the movie's matrix directly by calling the `SetMovieMatrix` function, which is described on page 2-170. You can retrieve a movie's boundary rectangle by calling the `GetMovieBox` function, which is described in the next section.

GetMovieBox

The `GetMovieBox` function returns a movie's boundary rectangle, which is a rectangle that encompasses all of the movie's enabled tracks. The movie box is in the coordinate system of the movie's graphics world and defines the movie's boundaries over the entire duration of the movie. The movie's boundary rectangle defines the size and shape of the movie before the Movie Toolbox applies the display clipping region.

```
pascal void GetMovieBox (Movie theMovie, Rect *boxRect);
```

`theMovie` Specifies the movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

`boxRect` Contains a pointer to a rectangle. The `GetMovieBox` function returns the coordinates of the movie's boundary rectangle into the structure referred to by this parameter.

ERROR CODES

`invalidMovie` -2010 This movie is corrupted or invalid
 Memory Manager errors

SEE ALSO

You can use the `SetMovieBox` function, which is described in the previous section, to change the coordinates of a movie's boundary rectangle.

GetMovieDisplayBoundsRgn

The `GetMovieDisplayBoundsRgn` function allows your application to determine a movie's display boundary region. The display boundary region encloses all of a movie's enabled tracks after the track matrix, track clip, movie matrix, and movie clip have been applied to all of the movie's tracks. This region is in the display coordinate system of the movie's graphics world. The movie's boundary rectangle encloses this region. For more on boundary regions and matrices for movies and tracks, see "Spatial Properties," which begins on page 2-20.

```
pascal RgnHandle GetMovieDisplayBoundsRgn (Movie theMovie);
```

theMovie Specifies the movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

DESCRIPTION

The Movie Toolbox derives the display boundary region only from enabled tracks, and only from those tracks that are used in the current display mode (that is, movie, poster, or preview). The display boundary region is valid for the current movie time.

The `GetMovieDisplayBoundsRgn` function allocates the region and returns a handle to the region. Your application must dispose of this handle when you are done with it. If the movie does not have a spatial representation at the current movie time, the function returns an empty region. If the function could not satisfy your request, it sets the returned handle to `nil`.

ERROR CODES

`invalidMovie` -2010 This movie is corrupted or invalid
Memory Manager errors

SEE ALSO

If you want to determine the boundary region that applies to a time segment of a movie, you can use the `GetMovieSegmentDisplayBoundsRegion` function, which is described in the next section.

GetMovieSegmentDisplayBoundsRgn

The `GetMovieSegmentDisplayBoundsRgn` function allows your application to determine a movie's display boundary region during a specified segment. The display boundary region encloses all of a movie's enabled tracks after the track matrix, track clip, movie matrix, and movie clip have been applied to all of the movie's tracks. This region is in the display coordinate system. The movie's boundary encloses this region. For more on boundary regions and matrices for movies and tracks, see "Spatial Properties," which begins on page 2-20.

```
pascal RgnHandle GetMovieSegmentDisplayBoundsRgn (Movie theMovie,
                                                    TimeValue time,
                                                    TimeValue
                                                    duration);
```

<code>theMovie</code>	Specifies the movie for this operation. Your application obtains this movie identifier from such functions as <code>NewMovie</code> , <code>NewMovieFromFile</code> , and <code>NewMovieFromHandle</code> (described on page 2-92, page 2-88, and page 2-90, respectively).
<code>time</code>	Specifies the starting time of the movie segment to consider. This time value must be expressed in the movie's time coordinate system. The <code>duration</code> parameter specifies the length of the segment.
<code>duration</code>	Specifies the length of the segment to consider. Set this parameter to 0 to specify an instant in time.

DESCRIPTION

The Movie Toolbox derives the display boundary region only from enabled tracks and only from those tracks that are used in the current display mode (that is, movie, poster, or preview). If you want to determine the boundary region that applies to the current movie time, you can use `GetMovieDisplayBoundsRegion`, which is described in the previous section.

The `GetMovieSegmentDisplayBoundsRgn` function allocates the region and returns a handle to the region. Your application must dispose of this region when you are done with it. If the movie does not have a spatial representation during the specified segment, the function returns an empty region. If the function could not satisfy your request, it sets the returned handle to `nil`.

ERROR CODES

<code>invalidMovie</code>	-2010	This movie is corrupted or invalid
<code>invalidDuration</code>	-2014	This duration value is invalid
<code>invalidTime</code>	-2015	This time value is invalid

Memory Manager errors

SetMovieDisplayClipRgn

The `SetMovieDisplayClipRgn` function allows your application to establish a movie's current display clipping region.

```
pascal void SetMovieDisplayClipRgn (Movie theMovie,
                                     RgnHandle theClip);
```

<code>theMovie</code>	Specifies the movie for this operation. Your application obtains this movie identifier from such functions as <code>NewMovie</code> , <code>NewMovieFromFile</code> , and <code>NewMovieFromHandle</code> (described on page 2-92, page 2-88, and page 2-90, respectively).
<code>theClip</code>	Contains a handle to the movie's display clipping region. Note that the Movie Toolbox makes a copy of this region. Your application must dispose of the region referred to by this parameter when you are done with it. Set this parameter to <code>nil</code> to disable a movie's clipping region.

DESCRIPTION

The display clipping region defines any final clipping that is applied to the movie before it is displayed, and it is valid for the entire duration of the movie. You must use this region to clip a movie because the Movie Toolbox ignores the clip region of the movie's graphics world during display processing.

Note that the display clipping region is not saved with the movie.

SPECIAL CONSIDERATIONS

Do not use the `SetMovieDisplayClipRgn` function when you are using a movie controller component—use the movie controller component function `MCSetClip` instead. For details on the `MCSetClip` function, see the chapter “Movie Controller Components” in *Inside Macintosh: QuickTime Components*.

ERROR CODES

<code>invalidMovie</code>	-2010	This movie is corrupted or invalid
Memory Manager errors		

SEE ALSO

You can retrieve the display clipping region by calling the `GetMovieDisplayClipRgn` function, which is described in the next section.

GetMovieDisplayClipRgn

The `GetMovieDisplayClipRgn` function allows your application to determine a movie's current display clipping region.

```
pascal RgnHandle GetMovieDisplayClipRgn (Movie theMovie);
```

theMovie Specifies the movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

DESCRIPTION

The display clipping region defines the final clipping that is applied to the movie before it is displayed. The display clipping region is valid for the entire duration of the movie.

Note that the display clipping region is not saved with the movie.

The `GetMovieDisplayClipRgn` function allocates the region and returns a handle to the region. Your application must dispose of this region when you are done with it. If the function could not satisfy your request or if there is no display clipping region defined for the movie, the function sets the returned handle to `nil`.

ERROR CODES

`invalidMovie` -2010 This movie is corrupted or invalid
Memory Manager errors

SEE ALSO

You can set the display clipping region by calling the `SetMovieDisplayClipRgn` function, which is described in the previous section.

GetTrackDisplayBoundsRgn

The `GetTrackDisplayBoundsRgn` function allows your application to determine the region a track occupies in a movie's graphics world.

```
pascal RgnHandle GetTrackDisplayBoundsRgn (Track theTrack);
```

theTrack Specifies the track for this operation. Your application obtains this track identifier from such Movie Toolbox functions as `NewMovieTrack` and `GetMovieTrack` (described on page 2-151 and page 2-204, respectively).

DESCRIPTION

This region is in the display coordinate system. This region, when intersected with the movie’s display clipping region, describes which pixels in the movie’s graphics world display information from the specified track. This region is valid for the current movie time.

The `GetTrackDisplayBoundsRgn` function allocates the region and returns a handle to the region. Your application must dispose of this region when you are done with it. If the track does not have a spatial representation at the current movie time, the function returns an empty region. If the function could not satisfy your request, it sets the returned handle to `nil`.

ERROR CODES

`invalidTrack` `-2009` This track is corrupted or invalid

Memory Manager errors

SEE ALSO

If you want to determine the track’s boundary region over a specified time segment, you can use the `GetTrackSegmentDisplayBoundsRgn` function, which is described in the next section.

GetTrackSegmentDisplayBoundsRgn

The `GetTrackSegmentDisplayBoundsRgn` function allows your application to determine the region a track occupies in a movie’s graphics world during a specified segment.

```
pascal RgnHandle GetTrackSegmentDisplayBoundsRgn (Track theTrack,
                                                    TimeValue time,
                                                    TimeValue duration);
```

- `theTrack` Specifies the track for this operation. Your application obtains this track identifier from such Movie Toolbox functions as `NewMovieTrack` and `GetMovieTrack` (described on page 2-151 and page 2-204, respectively).
- `time` Specifies the starting time of the track segment to consider. This time value must be expressed in the movie’s time coordinate system. The `duration` parameter specifies the length of the segment.
- `duration` Specifies the length of the segment to consider. Set this parameter to 0 to consider an instant in time.

Movie Toolbox

DESCRIPTION

This region is in the display coordinate system. When combined with the movie's display clipping region, this region describes which pixels in the movie's graphics world display information from the specified track.

This region is valid for the specified segment.

The `GetTrackSegmentDisplayBoundsRgn` function allocates the region and returns a handle to the region. Your application must dispose of this region when you are done with it. If the track does not have a spatial representation during the specified segment, the function returns an empty region. If the function could not satisfy your request, it sets the returned handle to `nil`.

ERROR CODES

<code>invalidTrack</code>	-2009	This track is corrupted or invalid
<code>invalidDuration</code>	-2014	This duration value is invalid
<code>invalidTime</code>	-2015	This time value is invalid

Memory Manager errors

SEE ALSO

If you want to determine the track's boundary region for the current movie time, you can use the `GetTrackDisplayBoundsRgn` function, which is described in the previous section.

SetTrackLayer

The `SetTrackLayer` function allows your application to set a track's layer.

```
pascal void SetTrackLayer (Track theTrack, short layer);
```

<code>theTrack</code>	Specifies the track for this operation. Your application obtains this track identifier from such Movie Toolbox functions as <code>NewMovieTrack</code> and <code>GetMovieTrack</code> (described on page 2-151 and page 2-204, respectively).
<code>layer</code>	Specifies the track's layer number. Layers are numbered from -32,768 through 32,767. When you create a new track, the Movie Toolbox sets its track number to 0.

DESCRIPTION

Track layers are numbered from -32,768 through 32,767. You can use layers to control how tracks are combined to create a movie. The Movie Toolbox displays layers by layer number. That is, the Movie Toolbox displays higher-numbered layers first, placing lower-numbered layers on top of them. If your movie has more than one track in the

same layer, the Movie Toolbox displays those layers in order by track index value, displaying higher-numbered tracks first.

ERROR CODES

<code>invalidTrack</code>	<code>-2009</code>	This track is corrupted or invalid
---------------------------	--------------------	------------------------------------

SEE ALSO

You can retrieve a track's layer number by calling the `GetTrackLayer` function, which is described in the next section.

GetTrackLayer

The `GetTrackLayer` function allows your application to retrieve a track's layer.

```
pascal short GetTrackLayer (Track theTrack);
```

<code>theTrack</code>	Specifies the track for this operation. Your application obtains this track identifier from such Movie Toolbox functions as <code>NewMovieTrack</code> and <code>GetMovieTrack</code> (described on page 2-151 and page 2-204, respectively).
-----------------------	---

DESCRIPTION

The `GetTrackLayer` function returns an integer that contains the track's layer number. Tracks are numbered from -32,768 through 32,767. You can use layers to control how tracks are combined to create a movie. The Movie Toolbox displays layers by layer number. That is, the Movie Toolbox displays higher-numbered layers first, placing lower-numbered layers on top of them. If your movie has more than one track in the same layer, the Movie Toolbox displays those layers in order by track index value, displaying higher-numbered tracks first.

ERROR CODES

<code>invalidTrack</code>	<code>-2009</code>	This track is corrupted or invalid
---------------------------	--------------------	------------------------------------

SEE ALSO

You can set a track's layer number by calling the `SetTrackLayer` function, which is described in the previous section.

SetMovieMatrix

The `SetMovieMatrix` function allows your application to set a movie's transformation matrix. The Movie Toolbox uses a movie's matrix to map a movie from its display coordinate system to its graphics world. You can retrieve a movie's matrix with the `GetMovieMatrix` function, which is described in the next section.

```
pascal void SetMovieMatrix (Movie theMovie,
                           const MatrixRecord *matrix);
```

<code>theMovie</code>	Specifies the movie for this operation. Your application obtains this movie identifier from such functions as <code>NewMovie</code> , <code>NewMovieFromFile</code> , and <code>NewMovieFromHandle</code> (described on page 2-92, page 2-88, and page 2-90, respectively).
<code>matrix</code>	Contains a pointer to the matrix structure for the movie. If you set this parameter to <code>nil</code> , the Movie Toolbox uses the identity matrix.

SPECIAL CONSIDERATIONS

The `SetMovieMatrix` function does not call your cover functions.

ERROR CODES

<code>invalidMovie</code>	-2010	This movie is corrupted or invalid
---------------------------	-------	------------------------------------

SEE ALSO

The Movie Toolbox provides a number of functions that allow you to manipulate movie matrices. See “Matrix Functions,” which begins on page 2-341, for information about these functions.

GetMovieMatrix

The `GetMovieMatrix` function allows your application to retrieve a movie's transformation matrix.

```
pascal void GetMovieMatrix (Movie theMovie, MatrixRecord *matrix);
```

<code>theMovie</code>	Specifies the movie for this operation. Your application obtains this movie identifier from such functions as <code>NewMovie</code> , <code>NewMovieFromFile</code> , and <code>NewMovieFromHandle</code> (described on page 2-92, page 2-88, and page 2-90, respectively).
<code>matrix</code>	Contains a pointer to a matrix structure. The <code>GetMovieMatrix</code> function returns the movie's matrix into the structure referred to by this parameter.

DESCRIPTION

The Movie Toolbox uses a movie's matrix to map a movie from its coordinate system to the display coordinate system.

ERROR CODES

`invalidMovie` -2010 This movie is corrupted or invalid

SEE ALSO

You can set a movie's matrix with the `SetMovieMatrix` function, which is described in the previous section.

The Movie Toolbox provides a number of functions that allow you to manipulate movie matrices. See "Matrix Functions," which begins on page 2-341, for information about these functions.

GetMovieBoundsRgn

The `GetMovieBoundsRgn` function allows your application to determine a movie's boundary region.

```
pascal RgnHandle GetMovieBoundsRgn (Movie theMovie);
```

`theMovie` Specifies the movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

DESCRIPTION

The movie boundary region encloses all of a movie's tracks after the union of the track clip and the track matrix has been applied to all the movie's tracks (but not to the movie itself). This region is in the movie's display coordinate system.

The Movie Toolbox derives the boundary region only from enabled tracks, and only from those tracks that are used in the current display mode (that is, movie or preview). The boundary region is valid for the current movie time.

The `GetMovieBoundsRgn` function allocates the region and returns a handle to the region. Your application must dispose of this region when you are done with it. If the movie does not have a spatial representation at the current time, the function returns an empty region. If the function could not satisfy your request, it sets the returned handle to `nil`.

Movie Toolbox

ERROR CODES

`invalidMovie` -2010 This movie is corrupted or invalid
 Memory Manager errors

GetTrackMovieBoundsRgn

The `GetTrackMovieBoundsRgn` function allows your application to determine the region the track occupies in a movie's boundary region. This region is in the display coordinate system of the movie. The Movie Toolbox determines this region by applying the track's clipping region and matrix. This region is valid only for the current movie time.

```
pascal RgnHandle GetTrackMovieBoundsRgn (Track theTrack);
```

`theTrack` Specifies the track for this operation. Your application obtains this track identifier from such Movie Toolbox functions as `NewMovieTrack` and `GetMovieTrack` (described on page 2-151 and page 2-204, respectively).

DESCRIPTION

The `GetTrackMovieBoundsRgn` function allocates the region and returns a handle to the region. Your application must dispose of this region when you are done with it. If the track does not have a spatial representation at the current movie time, the function returns an empty region. If the function could not satisfy your request, it sets the returned handle to `nil`.

ERROR CODES

`invalidTrack` -2009 This track is corrupted or invalid

SetMovieClipRgn

The `SetMovieClipRgn` function allows your application to establish a movie's clipping region.

```
pascal void SetMovieClipRgn (Movie theMovie, RgnHandle theClip);
```

`theMovie` Specifies the movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

Movie Toolbox

theClip Contains a handle to the movie's clipping region. Note that the Movie Toolbox makes a copy of this region. Your application must dispose of the region referred to by this parameter when you are done with it. Set this parameter to `nil` to disable clipping for the movie.

DESCRIPTION

The clipping region defines any clipping that is applied to the movie before it is mapped to its graphics world by applying the movie's matrix. The clipping region is in the movie's display coordinate system.

The clipping region is saved with the movie.

ERROR CODES

invalidMovie -2010 This movie is corrupted or invalid
Memory Manager errors

SEE ALSO

You can retrieve the clipping region by calling the `GetMovieClipRgn` function, which is described in the next section.

GetMovieClipRgn

The `GetMovieClipRgn` function allows your application to determine a movie's clipping region.

```
pascal RgnHandle GetMovieClipRgn (Movie theMovie);
```

theMovie Specifies the movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

DESCRIPTION

The clipping region defines any clipping that is applied to the movie before it is mapped to its graphics world by applying the movie's matrix. The clipping region is in the movie's display coordinate system and is valid for the entire duration of the movie.

The `GetMovieClipRgn` function allocates the region and returns a handle to the region. Your application must dispose of this region when you are done with it. If the function could not satisfy your request or if there is no clipping region defined for the movie, it sets the returned handle to `nil`.

The clipping region is saved with the movie when your application saves the movie.

Movie Toolbox

ERROR CODES

`invalidMovie` -2010 This movie is corrupted or invalid
 Memory Manager errors

SEE ALSO

You can set the clipping region by calling the `SetMovieClipRgn` function, which is described in the previous section.

SetTrackMatrix

The `SetTrackMatrix` function allows your application to establish a track's transformation matrix.

```
pascal void SetTrackMatrix (Track theTrack,
                             const MatrixRecord *matrix);
```

`theTrack` Specifies the track for this operation. Your application obtains this track identifier from such Movie Toolbox functions as `NewMovieTrack` and `GetMovieTrack` (described on page 2-151 and page 2-204, respectively).

`matrix` Contains a pointer to a matrix structure that contains the track's new matrix. If you set this parameter to `nil`, the Movie Toolbox uses the identity matrix.

DESCRIPTION

The Movie Toolbox uses a track's matrix to map a track from its own coordinate system into a movie's display coordinate system.

ERROR CODES

`invalidTrack` -2009 This track is corrupted or invalid

SEE ALSO

You can get a track's matrix with the `GetTrackMatrix` function, which is described in the next section.

The Movie Toolbox provides a number of functions that allow you to manipulate track matrices. See "Matrix Functions" beginning on page 2-341 for information about these functions.

GetTrackMatrix

The `GetTrackMatrix` function allows your application to retrieve a track's transformation matrix.

```
pascal void GetTrackMatrix (Track theTrack, MatrixRecord *matrix);
```

<code>theTrack</code>	Specifies the track for this operation. Your application obtains this track identifier from such Movie Toolbox functions as <code>NewMovieTrack</code> and <code>GetMovieTrack</code> (described on page 2-151 and page 2-204, respectively).
<code>matrix</code>	Contains a pointer to a matrix structure. The <code>GetTrackMatrix</code> function returns the track's matrix into the structure referred to by this parameter.

DESCRIPTION

The Movie Toolbox uses a track's matrix to map a track from its own coordinate system into a movie's display coordinate system.

ERROR CODES

<code>invalidTrack</code>	-2009	This track is corrupted or invalid
---------------------------	-------	------------------------------------

SEE ALSO

You can set a track's matrix with the `SetTrackMatrix` function, which is described in the previous section.

The Movie Toolbox provides a number of functions that allow you to manipulate track matrices. See "Matrix Functions" on page 2-341 for information about these functions.

GetTrackBoundsRgn

The `GetTrackBoundsRgn` function allows the media to limit the size of the track boundary rectangle. Therefore, the region returned by `GetTrackBoundsRgn` may not be rectangular and may be smaller than the track boundary region.

```
pascal RgnHandle GetTrackBoundsRgn (Track theTrack);
```

<code>theTrack</code>	Specifies the track for this operation. Your application obtains this track identifier from such Movie Toolbox functions as <code>NewMovieTrack</code> and <code>GetMovieTrack</code> (described on page 2-151 and page 2-204, respectively).
-----------------------	---

Movie Toolbox

DESCRIPTION

The `GetTrackBoundsRgn` function allocates the region and returns a handle to the region. Your application must dispose of this region when you are done with it. If the track does not have a spatial representation during the specified segment, the function returns an empty region. If the function could not satisfy your request, it sets the returned handle to `nil`.

ERROR CODES

`invalidTrack` -2009 This track is corrupted or invalid
Memory Manager errors

SEE ALSO

See the description of the base media handler component's `MediaGetSrcRgn` function in *Inside Macintosh: QuickTime Components* for details on how the media limits the size of the track boundary region.

SetTrackDimensions

The `SetTrackDimensions` function allows your application to establish a track's source, or display, rectangle.

```
pascal void SetTrackDimensions (Track theTrack, Fixed width,
                                Fixed height);
```

<code>theTrack</code>	Specifies the track for this operation. Your application obtains this track identifier from such Movie Toolbox functions as <code>NewMovieTrack</code> and <code>GetMovieTrack</code> (described on page 2-151 and page 2-204, respectively).
<code>width</code>	Contains a fixed-point number that specifies the width, in pixels, of the track's rectangle. This value corresponds to the x coordinate of the lower-right corner of the track's rectangle.
<code>height</code>	Contains a fixed-point number that specifies the height, in pixels, of the track's rectangle. This value corresponds to the y coordinate of the lower-right corner of the track's rectangle.

DESCRIPTION

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.

If you change the dimensions of an existing track, the media data is scaled to fit into the new rectangle.

ERROR CODES

`invalidTrack` `-2009` This track is corrupted or invalid

SEE ALSO

You can use the `GetTrackDimensions` function, which is described in the next section, to retrieve a track's rectangle.

GetTrackDimensions

The `GetTrackDimensions` function allows your application to determine a track's source, or display, rectangle.

```
pascal void GetTrackDimensions (Track theTrack, Fixed *width,
                                Fixed *height);
```

<code>theTrack</code>	Specifies the track for this operation. Your application obtains this track identifier from such Movie Toolbox functions as <code>NewMovieTrack</code> and <code>GetMovieTrack</code> (described on page 2-151 and page 2-204, respectively).
<code>width</code>	Contains a pointer to a fixed-point number. The Movie Toolbox returns the width, in pixels, of the track's rectangle. This value corresponds to the x coordinate of the lower-right corner of the track's rectangle.
<code>height</code>	Contains a pointer to a fixed-point number. The Movie Toolbox returns the height, in pixels, of the track's rectangle. This value corresponds to the y coordinate of the lower-right corner of the track's rectangle.

DESCRIPTION

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.

ERROR CODES

`invalidTrack` `-2009` This track is corrupted or invalid

SEE ALSO

You can use the `SetTrackDimensions` function, which is described in the previous section, to set a track's rectangle.

SetTrackClipRgn

The `SetTrackClipRgn` function allows your application to set the clipping region of a track.

```
pascal void SetTrackClipRgn (Track theTrack, RgnHandle theClip);
```

<code>theTrack</code>	Specifies the track for this operation. Your application obtains this track identifier from such Movie Toolbox functions as <code>NewMovieTrack</code> and <code>GetMovieTrack</code> (described on page 2-151 and page 2-204, respectively).
<code>theClip</code>	Contains a handle to the track's clipping region. Note that the Movie Toolbox makes a copy of this region. Your application must dispose of the region referred to by this parameter when you are done with it. Set this parameter to <code>nil</code> to disable clipping for the track.

DESCRIPTION

The clipping region is in the track's coordinate system. The Movie Toolbox applies the clipping region to a track before it applies the track's matrix.

ERROR CODES

<code>invalidTrack</code>	-2009	This track is corrupted or invalid
Memory Manager errors		

SEE ALSO

You can get a track's clipping region by calling the `GetTrackClipRgn` function, which is described in the next section.

GetTrackClipRgn

The `GetTrackClipRgn` function allows your application to determine the clipping region of a track.

```
pascal RgnHandle GetTrackClipRgn (Track theTrack);
```

<code>theTrack</code>	Specifies the track for this operation. Your application obtains this track identifier from such Movie Toolbox functions as <code>NewMovieTrack</code> and <code>GetMovieTrack</code> (described on page 2-151 and page 2-204, respectively).
-----------------------	---

DESCRIPTION

The clipping region is in the track's coordinate system. The Movie Toolbox applies the clipping region to a track before it applies the track's matrix. This region is valid for the entire duration of the track.

The `GetTrackClipRgn` function allocates the region and returns a handle to the region. Your application must dispose of this region when you are done with it. If the function could not satisfy your request or if there is no clipping region defined for the track, it sets the returned handle to `nil`.

ERROR CODES

`invalidTrack` -2009 This track is corrupted or invalid
Memory Manager errors

SEE ALSO

You can establish a track's clipping region by calling the `SetTrackClipRgn` function, which is described in the previous section.

SetTrackMatte

The `SetTrackMatte` function allows your application to set a track's matte. The matte defines which of the track's pixels are displayed in a movie. You must specify the matte in a pixel map structure.

```
pascal void SetTrackMatte (Track theTrack, PixMapHandle theMatte);
```

`theTrack` Specifies the track for this operation. Your application obtains this track identifier from such Movie Toolbox functions as `NewMovieTrack` and `GetMovieTrack` (described on page 2-151 and page 2-204, respectively).

`theMatte` Contains a handle to the matte. The Movie Toolbox makes a copy of the matte, including its color table and pixels. Consequently, your application must dispose of the matte when you are done with it. Set this parameter to `nil` to remove the track's matte.

DESCRIPTION

The Movie Toolbox displays the weighted average of the track and its destination based on the corresponding pixel in the matte (this feature is fully functional in System 7 and is approximated in System 6).

SPECIAL CONSIDERATIONS

Note that the track matte must have its boundaries defined by the track rectangle.

Movie Toolbox

ERROR CODES

`invalidTrack` -2009 This track is corrupted or invalid
Memory Manager errors

SEE ALSO

You can retrieve a track's matte by calling the `GetTrackMatte` function, which is described in the next section. Listing 2-15 on page 2-73 shows how to use the `SetTrackMatte` and `GetTrackMatte` functions to create a track matte.

GetTrackMatte

The `GetTrackMatte` function allows your application to retrieve a copy of a track's matte. The matte defines which of the track's pixels are displayed in a movie, and it is valid for the entire duration of the movie. This function returns the matte in a pixel map structure. You may use QuickDraw functions to manipulate the returned matte. However, you should use the Movie Toolbox's `DisposeMatte` function (described in the next section) to dispose of the matte when you are finished with it.

```
pascal PixmapHandle GetTrackMatte (Track theTrack);
```

`theTrack` Specifies the track for this operation. Your application obtains this track identifier from such Movie Toolbox functions as `NewMovieTrack` and `GetMovieTrack` (described on page 2-151 and page 2-204, respectively).

DESCRIPTION

The `GetTrackMatte` function returns a handle to the matte. Your application must dispose of this handle when you are done with it—you must use the `DisposeMatte` function, which is described in the next section, to dispose of the matte. If the function could not satisfy your request, it sets the returned handle to `nil`.

ERROR CODES

`invalidTrack` -2009 This track is corrupted or invalid
Memory Manager errors

SEE ALSO

You can establish a track's matte by calling the `SetTrackMatte` function, which is described in the previous section. Listing 2-15 on page 2-73 shows how to use the `SetTrackMatte` and `GetTrackMatte` functions to create a track matte.

DisposeMatte

The `DisposeMatte` function disposes of a matte that you obtained from the `GetTrackMatte` function, which is described in the previous section.

```
pascal void DisposeMatte (PixMapHandle theMatte);
```

theMatte Handle to the matte to be disposed. Your application obtains this handle from the `GetTrackMatte` function.

SPECIAL CONSIDERATIONS

You should not use this function to dispose of mattes or pixel maps that you obtain through other means.

ERROR CODES

None

Working With Sound Volume

The Movie Toolbox allows you to set the sound volume of movies and tracks. Track volumes allow tracks within a movie to have different volumes. A track's volume is scaled by the movie's volume to produce the track's final volume. Furthermore, the movie's volume is scaled by the sound volume that is returned by the Sound Manager's `GetSoundVol` routine. Thus, the user can control the overall volume from the Sound control panel.

Volume values range from -1.0 to 1.0. Higher values translate to louder volume. Negative values indicate muted volume. That is, the Movie Toolbox does not play any sound for movies or tracks with negative volume settings, but the original volume level is retained as the absolute value of the volume setting. Therefore, if you want to toggle the current state of the volume, you can invert the sign of the current volume setting, as shown here:

```
SetMovieVolume (theMovie, -GetMovieVolume (theMovie));
```

You can use the `GetMovieVolume` and `SetMovieVolume` functions to work with a movie's volume.

The `GetTrackVolume` and `SetTrackVolume` functions allow you to work with a track's volume.

SetMovieVolume

The `SetMovieVolume` function allows your application to set a movie's current volume.

```
pascal void SetMovieVolume (Movie theMovie, short volume);
```

theMovie Specifies the movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

volume Specifies the current volume setting of the movie represented as a 16-bit, fixed-point number. The high-order 8 bits contain the integer part of the value; the low-order 8 bits contain the fractional part. Volume values range from -1.0 to 1.0. Negative values play no sound but preserve the absolute value of the volume setting.

`kFullVolume`

Sets the movie to full volume (constant value is 1.0).

`kNoVolume`

Sets the movie to no volume (constant value is 0.0).

ERROR CODES

`invalidMovie` -2010 This movie is corrupted or invalid

SEE ALSO

Your application can obtain the current volume setting by calling the `GetMovieVolume` function, which is described in the next section.

GetMovieVolume

The `GetMovieVolume` function returns a movie's current volume setting.

```
pascal short GetMovieVolume (Movie theMovie);
```

theMovie Specifies the movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

DESCRIPTION

The `GetMovieVolume` function returns an integer that contains the movie's current volume represented as a 16-bit, fixed-point number. The high-order 8 bits contain the integer part of the value; the low-order 8 bits contain the fractional part. Volume values

range from -1.0 to 1.0. Negative values play no sound but preserve the absolute value of the volume setting.

ERROR CODES

invalidMovie -2010 This movie is corrupted or invalid

SEE ALSO

You can change a movie’s current volume by calling the `SetMovieVolume` function, which is described in the previous section.

SetTrackVolume

The `SetTrackVolume` function allows your application to set a track’s current volume.

pascal void SetTrackVolume (Track theTrack, short volume);

- theTrack Specifies the track for this operation. Your application obtains this track identifier from such Movie Toolbox functions as `NewMovieTrack` and `GetMovieTrack` (described on page 2-151 and page 2-204, respectively).
- volume Specifies the current volume setting of the track represented as a 16-bit, fixed-point number. The high-order 8 bits contain the integer part of the value; the low-order 8 bits contain the fractional part. Volume values range from -1.0 to 1.0. Negative values play no sound but preserve the absolute value of the volume setting.
 - kFullVolume Sets the track to full volume (constant value is 1.0).
 - kNoVolume Sets the track to no volume (constant value is 0.0).

DESCRIPTION

Note that, when the track is played, the track’s volume is scaled by the volume setting of the movie that contains the track.

ERROR CODES

invalidTrack -2009 This track is corrupted or invalid

SEE ALSO

Your application can obtain the current volume setting by calling the `GetTrackVolume` function, which is described in the next section.

GetTrackVolume

The `GetTrackVolume` function returns a track's current volume setting.

```
pascal short GetTrackVolume (Track theTrack);
```

theTrack Specifies the track for this operation. Your application obtains this track identifier from such Movie Toolbox functions as `NewMovieTrack` and `GetMovieTrack` (described on page 2-151 and page 2-204, respectively).

DESCRIPTION

The `GetTrackVolume` function returns an integer that contains the track's current volume represented as a 16-bit, fixed-point number. The high-order 8 bits contain the integer part of the value; the low-order 8 bits contain the fractional part. Volume values range from -1.0 to 1.0. Negative values play no sound but preserve the absolute value of the volume setting.

ERROR CODES

`invalidTrack` -2009 This track is corrupted or invalid

SEE ALSO

You can change a track's current volume by calling the `SetTrackVolume` function, which is described in the previous section.

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. This section discusses 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 a complete discussion of the relationships between movie, track, and media time parameters, see "Introduction to Movies" beginning on page 2-5. For information about more functions that work with time, see "Time Base Functions" beginning on page 2-315.

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.

Movie Toolbox

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.

GetMovieDuration

The `GetMovieDuration` function returns the duration of a movie. The Movie Toolbox examines the durations of all the tracks of the movie to determine this value.

```
pascal TimeValue GetMovieDuration (Movie theMovie);
```

theMovie Specifies the movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

DESCRIPTION

The `GetMovieDuration` function returns a time value. This time value indicates the movie's duration, and it is expressed in the movie's time scale.

You cannot set movie direction explicitly because it is calculated as being the maximum durations of all the tracks in the movie.

ERROR CODES

<code>invalidMovie</code>	-2010	This movie is corrupted or invalid
---------------------------	-------	------------------------------------

SetMovieTimeValue

The `SetMovieTimeValue` function allows your application to set a movie's time value. You specify the new time as a time value, rather than in a time structure. You must ensure that the time value is in the movie's time scale.

```
pascal void SetMovieTimeValue (Movie theMovie, TimeValue newTime);
```

theMovie Specifies the movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

Movie Toolbox

newTime Specifies the movie's new time value. The Movie Toolbox interprets this time value relative to the movie's time scale. If you specify a value that is outside the duration of the movie, the Movie Toolbox sets the movie time to the beginning or end of the movie, as appropriate.

ERROR CODES

invalidMovie -2010 This movie is corrupted or invalid

SEE ALSO

You can also set a movie's current time by calling the `SetMovieTime` function, which is described in the next section. This function requires that you specify the new time value in a time structure.

SetMovieTime

The `SetMovieTime` function allows your application to change a movie's current time. You must specify the new time in a time structure. The Movie Toolbox saves the movie's current time when you save the movie.

```
pascal void SetMovieTime (Movie theMovie,
                          const TimeRecord *newTime);
```

theMovie Specifies the movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

newTime Contains a pointer to a time structure. If you specify a value that is outside the duration of the movie, the Movie Toolbox sets the movie time to the beginning or end of the movie, as appropriate.

ERROR CODES

invalidMovie -2010 This movie is corrupted or invalid

SEE ALSO

You can use the `SetMovieTimeValue` function, described in the previous section, to change a movie's current time without specifying a time structure.

You can retrieve a movie's current time value by calling the `GetMovieTime` function, which is described in the next section.

GetMovieTime

The `GetMovieTime` function returns a movie's current time. This function returns the time in two formats: as a time value and in a time structure.

```
pascal TimeValue GetMovieTime (Movie theMovie,
                                TimeRecord *currentTime);
```

theMovie Specifies the movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

currentTime Contains a pointer to a time structure. The `GetMovieTime` function updates this time structure to contain the movie's current time. If you do not want this information, set this parameter to `nil`.

DESCRIPTION

The `GetMovieTime` function returns a time value. This time value indicates the movie's current time, and it is expressed in the movie's time scale.

ERROR CODES

`invalidMovie` -2010 This movie is corrupted or invalid

SEE ALSO

You can set a movie's current time by calling the `SetMovieTime` or `SetMovieTimeValue` functions, which are described on page 2-186 and page 2-185, respectively.

SetMovieRate

The `SetMovieRate` function sets a movie's playback rate.

```
pascal void SetMovieRate (Movie theMovie, Fixed rate);
```

theMovie Specifies the movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

rate Specifies the new movie rate as a 32-bit, fixed-point number. Positive integers indicate forward rates and negative integers indicate reverse rates.

Movie Toolbox

ERROR CODES

`invalidMovie` -2010 This movie is corrupted or invalid

SEE ALSO

Your application can retrieve a movie's current playback rate by calling the `GetMovieRate` function, which is described in the next section. To play a movie at the movie's preferred rate from a position stored within the movie, you can use the `StartMovie` function (described on page 2-111).

GetMovieRate

The `GetMovieRate` function returns a movie's playback rate.

```
pascal Fixed GetMovieRate (Movie theMovie);
```

`theMovie` Specifies the movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

DESCRIPTION

The `GetMovieRate` function returns the movie rate as a 32-bit, fixed-point number. Positive integers indicate forward rates and negative integers indicate reverse rates.

ERROR CODES

`invalidMovie` -2010 This movie is corrupted or invalid

SEE ALSO

Your application can set the movie's playback rate by calling the `SetMovieRate` function, which is described in the previous section.

SetMovieTimeScale

The `SetMovieTimeScale` function establishes a movie's time scale.

```
pascal void SetMovieTimeScale (Movie theMovie,  
                               TimeScale timeScale);
```

theMovie Specifies the movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

timeScale Specifies the movie's new time scale.

DESCRIPTION

In response to this request, the Movie Toolbox adjusts the edit list of the movie's tracks so that movie playback is unaffected. If you change a movie's time scale by setting it to a smaller value (thereby losing precision in the movie's time values), the Movie Toolbox may edit information from the movie. In general, you should only increase the time scale value, and you should try to use integer multiples of the existing time scale.

SPECIAL CONSIDERATIONS

Do not call `SetMovieTimeScale` if you have edited your movie. This function quantizes the beginning and the end of the edits to the new units. Therefore, if you do not use an integral multiple, the position of your edits may change.

ERROR CODES

`invalidMovie` -2010 This movie is corrupted or invalid

SEE ALSO

You can retrieve a movie's time scale by calling the `GetMovieTimeScale` function, which is described in the next section.

GetMovieTimeScale

The `GetMovieTimeScale` function returns the time scale of a movie.

```
pascal TimeScale GetMovieTimeScale (Movie theMovie);
```

theMovie Specifies the movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

DESCRIPTION

The default QuickTime movie time scale is 600 units per second; however, this number may change in the future. The default time scale was chosen because it is convenient for working with common video frame rates of 30, 25, 24, 15, 12, 10, and 8.

ERROR CODES

`invalidMovie` -2010 This movie is corrupted or invalid

SEE ALSO

You can set a movie's time scale by calling the `SetMovieTimeScale` function, which is described in the previous section.

GetMovieTimeBase

The `GetMovieTimeBase` function returns a movie's time base.

```
pascal TimeBase GetMovieTimeBase (Movie theMovie);
```

theMovie Specifies the movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

DESCRIPTION

You cannot use the returned time base value with the Movie Toolbox's `SetTimeBaseMasterTimeBase` and `SetTimeBaseMasterClock` functions (described on page 2-320 and page 2-318, respectively). Use the `SetMovieMasterTimeBase` and `SetMovieMasterClock` functions (described on page 2-318 and page 2-317, respectively) instead.

The Movie Toolbox disposes of a movie's time base when you dispose of the movie.

SPECIAL CONSIDERATIONS

Do not dispose of the `TimeBase` result returned by the `GetMovieTimeBase` function as it is owned by the movie.

ERROR CODES

`invalidMovie` `-2010` This movie is corrupted or invalid

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. See Figure 2-6 on page 2-12 for a visual representation of track duration and track offset.

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.

GetTrackDuration

The `GetTrackDuration` function returns the duration of a track. The duration corresponds to the ending time of the track in the movie's time coordinate system (remember that all tracks start at movie time 0).

```
pascal TimeValue GetTrackDuration (Track theTrack);
```

theTrack Specifies the track for this operation. Your application obtains this track identifier from such Movie Toolbox functions as `NewMovieTrack` and `GetMovieTrack` (described on page 2-151 and page 2-204, respectively).

Movie Toolbox

DESCRIPTION

The `GetTrackDuration` function returns a time value. This time value indicates the track's duration, and it is expressed in the time scale of the movie that contains the track.

ERROR CODES

<code>invalidTrack</code>	-2009	This track is corrupted or invalid
---------------------------	-------	------------------------------------

SetTrackOffset

The `SetTrackOffset` function modifies the duration of the empty space that lies at the beginning of the track, thus changing the duration of the entire track. You specify this time offset as a time value in the movie's time scale. See Figure 2-6 on page 2-12 for an illustration of a track offset in a movie.

```
pascal void SetTrackOffset (Track theTrack,
                           TimeValue movieOffsetTime);
```

`theTrack` Specifies the track for this operation. Your application obtains this track identifier from such Movie Toolbox functions as `NewMovieTrack` and `GetMovieTrack` (described on page 2-151 and page 2-204, respectively).

`movieOffsetTime` Specifies the track's offset from the start of the movie, and must be expressed in the time scale of the movie that contains the track.

ERROR CODES

<code>invalidTrack</code>	-2009	This track is corrupted or invalid
---------------------------	-------	------------------------------------

SEE ALSO

You can determine a track's time offset by calling the `GetTrackOffset` function, which is described in the next section.

GetTrackOffset

The `GetTrackOffset` function allows your application to determine the time difference between the start of a track and the start of the movie that contains the track.

```
pascal TimeValue GetTrackOffset (Track theTrack);
```

theTrack Specifies the track for this operation. Your application obtains this track identifier from such Movie Toolbox functions as `NewMovieTrack` and `GetMovieTrack` (described on page 2-151 and page 2-204, respectively).

DESCRIPTION

The `GetTrackOffset` function returns a time value. This time value indicates the track's offset from the start of the movie, and it is expressed in the time scale of the movie that contains the track.

ERROR CODES

`invalidTrack` -2009 This track is corrupted or invalid

SEE ALSO

You can set a track's offset by calling the `SetTrackOffset` function, which is described in the previous section.

TrackTimeToMediaTime

The `TrackTimeToMediaTime` function allows your application to convert a track's time value to a time value that is appropriate to the track's media using the track's edit list. You specify the track's time in the movie's time coordinate system.

```
pascal TimeValue TrackTimeToMediaTime (TimeValue value,
                                         Track theTrack);
```

value Specifies the track's time value; must be expressed in the time scale of the movie that contains the track.

theTrack Specifies the track for this operation. Your application obtains this track identifier from such Movie Toolbox functions as `NewMovieTrack` and `GetMovieTrack` (described on page 2-151 and page 2-204, respectively).

Movie Toolbox

DESCRIPTION

The Movie Toolbox returns a value that is in the media's time coordinate system.

You can use the `TrackTimeToMediaTime` function to determine whether a specified track edit is empty. If the track time corresponds to empty space, this function returns a value of `-1`.

The `TrackTimeToMediaTime` function maps the track time through the track's edit list to come up with the media time. This time value contains the track's time value according to the media's time coordinate system. If the time you specified lies outside of the movie's active segment or corresponds to empty space in the track, the `TrackTimeToMediaTime` function returns a value of `-1`.

ERROR CODES

<code>invalidTrack</code>	<code>-2009</code>	This track is corrupted or invalid
---------------------------	--------------------	------------------------------------

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.

GetMediaDuration

The `GetMediaDuration` function returns the duration of a media.

```
pascal TimeValue GetMediaDuration (Media theMedia);
```

<code>theMedia</code>	Specifies the media for this operation. Your application obtains this media identifier from such Movie Toolbox functions as <code>NewTrackMedia</code> and <code>GetTrackMedia</code> (described on page 2-153 and page 2-206, respectively).
-----------------------	---

DESCRIPTION

The `GetMediaDuration` function returns a time value. This time value indicates the media's duration, and it is expressed in the time scale of the media.

ERROR CODES

<code>invalidMedia</code>	<code>-2008</code>	This media is corrupted or invalid
---------------------------	--------------------	------------------------------------

SetMediaTimeScale

The `SetMediaTimeScale` function allows your application to set a media's time scale.

```
pascal void SetMediaTimeScale (Media theMedia,
                               TimeScale timeScale);
```

theMedia Specifies the media for this operation. Your application obtains this media identifier from such Movie Toolbox functions as `NewTrackMedia` and `GetTrackMedia` (described on page 2-153 and page 2-206, respectively).

timeScale Specifies the media's new time scale.

DESCRIPTION

In response to this request, the Movie Toolbox attempts to adjust the edit list of the appropriate track so that movie playback is unaffected. If you change a media's time scale by setting it to a smaller value, you may lose precision in media time values. In general, you should only increase the time scale value, and you should try to use integer multiples of the existing time scale.

SPECIAL CONSIDERATIONS

Do not use `SetMediaTimeScale` as a general rule. If you call this function with a number that is not an integer multiple, the duration of the samples vary unpredictably, and their start times tend to drift.

ERROR CODES

`invalidMedia` -2008 This media is corrupted or invalid

GetMediaTimeScale

The `GetMediaTimeScale` function allows your application to determine a media's time scale.

```
pascal TimeScale GetMediaTimeScale (Media theMedia);
```

theMedia Specifies the media for this operation. Your application obtains this media identifier from such Movie Toolbox functions as `NewTrackMedia` and `GetTrackMedia` (described on page 2-153 and page 2-206, respectively).

Movie Toolbox

DESCRIPTION

The `GetMediaTimeScale` function returns the media's time scale.

ERROR CODES

`invalidMedia` -2008 This media is corrupted or invalid

Finding Interesting Times

The Movie Toolbox provides a set of functions that help you locate samples in movies, tracks, and media structures. These functions are based on the concept of “interesting times.” An interesting time refers to a time value in a movie, track, or media that meets certain search criteria. You specify the search criteria to the Movie Toolbox. The Movie Toolbox then scans the movie, track, or media, and locates time values that meet those search criteria.

You can use these functions to search through image sequences. For example, you may want to locate each frame in an image sequence. Or you may be more interested in key frames, especially if you are trying to optimize display performance. In image data, sync samples are referred to as **key frames**. For more information on key frames, see the chapter “Image Compression Manager” in this book. An easy way to determine whether a movie has been edited is to look for track edits in the movie data. You may also be interested in searching for samples in a movie's media. If you set the appropriate search criteria, the Movie Toolbox locates the appropriate frames for you. You need the functions described in this section because QuickTime doesn't have a fixed rate. Each frame can have its own duration.

The Movie Toolbox identifies an interesting time by specifying its starting time and duration. The starting time indicates the time in the movie, track, or media where the search criteria are met. The duration indicates the length of time during which the search criteria remain in effect. For example, if you are looking for samples in a media, the start time would indicate the beginning of the sample, and the duration would indicate the length of time to the next sample. In this case, you could find the next media sample by adding the duration to the start time. These duration values are always positive—you determine the direction of the search by setting the sign of the rate value you supply to the functions.

Note that movie interesting times are defined in the scope of the movie as a whole. As a result, one interesting time ends when another interesting time starts in any track in the movie. For example, if you are looking for key frames in a movie, the duration value from one interesting time tells you when the next key frame starts. However, that second key frame may be in a different track in the movie. Therefore, the duration of the interesting time does not necessarily correspond to the duration of the key frame.

You can use the `GetMovieNextInterestingTime` function to locate times of interest in a movie. The `GetTrackNextInterestingTime` function lets you work with tracks. Use the `GetMediaNextInterestingTime` function to locate samples in a media.

GetMovieNextInterestingTime

The `GetMovieNextInterestingTime` function searches for times of interest in a movie. This function examines only the movie's enabled tracks.

```
pascal void GetMovieNextInterestingTime (Movie theMovie,
                                         short interestingTimeFlags,
                                         short numMediaTypes,
                                         const OSType *whichMediaTypes,
                                         TimeValue time, Fixed rate,
                                         TimeValue *interestingTime,
                                         TimeValue *interestingDuration);
```

theMovie Specifies the movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

interestingTimeFlags

Specifies the search criteria. Note that you may set only one of the `nextTimeMediaSample`, `nextTimeMediaEdit`, `nextTimeTrackEdit` and `nextTimeSyncSample` flags to 1. The following flags are available (set unused flags to 0):

nextTimeMediaSample

Searches for the next sample in the movie's media. Set this flag to 1 to search for the next sample.

nextTimeMediaEdit

Searches for the next group of samples in the movie's media. Set this flag to 1 to search for the next group of samples.

nextTimeTrackEdit

Searches for the media sample that corresponds to the next entry in a track's media edit list. The end of the track is considered an empty edit. Set this flag to 1 to search for the next track edit.

nextTimeSyncSample

Searches for the next sync sample in the movie's media. Set this flag to 1 to search for the next sync sample.

Sync samples do not rely on preceding frames for content. Some compression algorithms conserve space by eliminating duplication between consecutive frames in a sample.

Movie Toolbox

`nextTimeEdgeOK`

Instructs the Movie Toolbox that you are willing to receive information about elements that begin or end at the time specified by the `time` parameter. Set this flag to 1 to accept this information.

This flag is especially useful at the beginning or end of a movie. The function returns valid information about the beginning and end of the movie.

`nextTimeIgnoreActiveSegment`

Instructs the Movie Toolbox to look outside of the active segment for samples that meet the search criteria. Set this flag to 1 to search outside of the active segment.

`numMediaTypes`

Specifies the number of media types in the table referred to by the `whichMediaType` parameter. Set this parameter to 0 to search all media types.

`whichMediaTypes`

Contains a pointer to an array of media types. You can use this parameter to limit the search to a specified set of media types. Each entry in the table referred to by this parameter identifies a media type to be included in the search. You use the `numMediaTypes` parameter to indicate the number of entries in the table. Set this parameter to `nil` to search all media types.

`VisualMediaCharacteristic 'eyes'`

Instructs the Movie Toolbox to search all tracks that have spatial bounds.

`AudioMediaCharacteristic 'ears'`

Instructs the Movie Toolbox to search all tracks that play sound.

`time`

Specifies a time value that establishes the starting point for the search. This time value must be expressed in the movie's time scale.

`rate`

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

`interestingTime`

Contains a pointer to a time value. The Movie Toolbox returns the first time value it finds that meets the search criteria specified in the `flags` parameter. This time value is in the movie's time scale.

If there are no times that meet the search criteria you specify, the Movie Toolbox sets this value to `-1`.

If you are not interested in this information, set this parameter to `nil`.

`interestingDuration`

Contains a pointer to a time value. The Movie Toolbox returns the duration of the interesting time. This time value is in the movie's time coordinate system. Set this parameter to `nil` if you do not want this information—in this case, the function works more quickly.

DESCRIPTION

You can use the `GetMovieNextInterestingTime` function to step through the frames of a movie one by one. If no tracks match the media types, the `invalidMedia` error is returned.

ERROR CODES

<code>invalidMedia</code>	-2008	This media is corrupted or invalid
<code>invalidTrack</code>	-2009	This track is corrupted or invalid
<code>invalidMovie</code>	-2010	This movie is corrupted or invalid
<code>invalidTime</code>	-2015	This time value is invalid

GetTrackNextInterestingTime

The `GetTrackNextInterestingTime` function searches for times of interest in a track.

```
pascal void GetTrackNextInterestingTime (Track theTrack,
                                         short interestingTimeFlags,
                                         TimeValue time, Fixed rate,
                                         TimeValue *interestingTime,
                                         TimeValue *interestingDuration);
```

theTrack Specifies the track for this operation. Your application obtains this track identifier from such Movie Toolbox functions as `NewMovieTrack` and `GetMovieTrack` (described on page 2-151 and page 2-204, respectively).

interestingTimeFlags Specifies the search criteria. Note that you may set only one of the `nextTimeMediaSample`, `nextTimeMediaEdit`, `nextTimeTrackEdit` and `nextTimeSyncSample` flags to 1. The following flags are available (set unused flags to 0):

nextTimeMediaSample

Searches for the next sample in the track's media. Set this flag to 1 to search for the next sample.

nextTimeMediaEdit

Searches for the next group of samples in the track's media. Set this flag to 1 to search for the next group of samples.

nextTimeTrackEdit

Searches for the media sample that corresponds to the next entry in a track's media edit list. The end of the track is considered an empty edit. Set this flag to 1 to search for the next track edit.

Movie Toolbox

`nextTimeSyncSample`

Searches for the next sync sample in the track's media. Set this flag to 1 to search for the next sync sample.

Sync samples do not rely on preceding frames for content. Some compression algorithms conserve space by eliminating duplication between consecutive frames in a sample.

`nextTimeEdgeOK`

Instructs the Movie Toolbox that you are willing to receive information about elements that begin or end at the time specified by the `time` parameter. Set this flag to 1 to accept this information.

This flag is especially useful at the beginning or end of a track. The function returns valid information about the beginning and end of the track.

`nextTimeIgnoreActiveSegment`

Instructs the Movie Toolbox to look outside of the active segment for samples that meet the search criteria. Set this flag to 1 to search outside of the active segment.

`time` Specifies a time value that establishes the starting point for the search. This time value must be expressed in the movie's time scale.

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

`interestingTime`

Contains a pointer to a time value. The Movie Toolbox returns the first time value it finds that meets the search criteria specified in the `flags` parameter. This time value is in the movie's time scale.

If there are no times that meet the search criteria you specify, the Movie Toolbox sets this value to -1.

Set this parameter to `nil` if you are not interested in this information.

`interestingDuration`

Contains a pointer to a time value. The Movie Toolbox returns the duration of the interesting time. This time value is in the movie's time coordinate system. Set this parameter to `nil` if you do not want this information—in this case, the function works more quickly.

ERROR CODES

<code>invalidTrack</code>	-2009	This track is corrupted or invalid
<code>invalidTime</code>	-2015	This time value is invalid

GetMediaNextInterestingTime

The `GetMediaNextInterestingTime` function searches for times of interest in a media.

```
pascal void GetMediaNextInterestingTime (Media theMedia,
                                         short interestingTimeFlags,
                                         TimeValue time, Fixed rate,
                                         TimeValue *interestingTime,
                                         TimeValue *interestingDuration);
```

theMedia Specifies the media for this operation. Your application obtains this media identifier from such Movie Toolbox functions as `NewTrackMedia` and `GetTrackMedia` (described on page 2-153 and page 2-206, respectively).

interestingTimeFlags Specifies the search criteria. Note that you may set only one of the `nextTimeMediaSample`, `nextTimeMediaEdit` and `nextTimeSyncSample` flags to 1. The following flags are available (set unused flags to 0):

`nextTimeMediaSample`

Searches for the next sample in the media. Set this flag to 1 to search for the next sample.

`nextTimeMediaEdit`

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

`nextTimeSyncSample`

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

Sync samples do not rely on preceding frames for content. Some compression algorithms conserve space by eliminating duplication between consecutive frames in a sample.

`nextTimeEdgeOK`

Instructs the Movie Toolbox that you are willing to receive information about elements that begin or end at the time specified by the `time` parameter. Set this flag to 1 to accept this information.

This flag is especially useful at the beginning or end of a media. The function returns valid information about the beginning and end of the media.

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

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

Movie Toolbox

`interestingTime`

Contains a pointer to a time value. The Movie Toolbox returns the first time value it finds that meets the search criteria specified in the `flags` parameter. This time value is in the media's time scale.

If there are no times that meet the search criteria you specify, the Movie Toolbox sets this value to `-1`.

Set this parameter to `nil` if you are not interested in this information.

`interestingDuration`

Contains a pointer to a time value. The Movie Toolbox returns the duration of the interesting time. This time value is in the media's time coordinate system. Set this parameter to `nil` if you do not want this information—in this case, the function works more quickly.

DESCRIPTION

`GetMediaNextInterestingTime` ignores all the edits that are defined in a movie's tracks.

ERROR CODES

<code>invalidMedia</code>	<code>-2008</code>	This media is corrupted or invalid
<code>invalidTime</code>	<code>-2015</code>	This time value is invalid

Locating a Movie's Tracks and Media Structures

The Movie Toolbox provides a set of functions that help your application locate a movie's tracks and media structures. This section describes these functions.

The Movie Toolbox identifies a movie's tracks in two ways. First, every track in a movie has a unique ID value. This ID value is unique throughout the life of a movie, even after it has been saved. That is, no two tracks of a movie ever have the same ID, and no ID value is ever reused. Second, a movie's current tracks may be identified by their index value. Index values always range from 1 to the number of tracks in the movie. Track indexes provide a convenient way to access each track of a movie.

There are several functions that allow you to find a movie's tracks. You can use the `GetMovieTrackCount` function to determine the number of tracks in a movie. Use the `GetMovieTrack` function to obtain the track identifier for a specific track, given its ID. The `GetMovieIndTrack` function lets you obtain a track's identifier, given its track index.

You can obtain a track's ID value given its track identifier by calling the `GetTrackID` function.

You can determine the movie that contains a track by calling the `GetTrackMovie` function.

The `GetTrackMedia` function enables you to find a track's media. Conversely, you can find the track that uses a media by calling the `GetMediaTrack` function.

GetMovieTrackCount

The `GetMovieTrackCount` function returns the number of tracks in a movie.

```
pascal long GetMovieTrackCount (Movie theMovie);
```

theMovie Specifies the movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

ERROR CODES

`invalidMovie` -2010 This movie is corrupted or invalid

GetMovieIndTrack

The `GetMovieIndTrack` function allows your application to determine the track identifier of a track given the track's index value. The index value identifies the track among all current tracks in a movie. Index values range from 1 to the number of tracks in the movie.

```
pascal Track GetMovieIndTrack (Movie theMovie, long index);
```

theMovie Specifies the movie for this operation. Your application obtains this movie identifier from such functions as `NewMovie`, `NewMovieFromFile`, and `NewMovieFromHandle` (described on page 2-92, page 2-88, and page 2-90, respectively).

index Specifies the index value of the track for this operation.

DESCRIPTION

The `GetMovieIndTrack` function returns the track identifier that is appropriate to the specified track. If the function cannot locate the track, it sets this returned value to `nil`.

Movie Toolbox

ERROR CODES

<code>badTrackIndex</code>	-2028	This track index value is not valid
<code>invalidMovie</code>	-2010	This movie is corrupted or invalid

SEE ALSO

You can determine the number of tracks in a movie by calling the `GetMovieTrackCount` function, which is described in the previous section.

GetMovieTrack

The `GetMovieTrack` function allows your application to determine the track identifier of a track given the track's ID value.

```
pascal Track GetMovieTrack (Movie theMovie, long trackID);
```

<code>theMovie</code>	Specifies the movie for this operation. Your application obtains this movie identifier from such functions as <code>NewMovie</code> , <code>NewMovieFromFile</code> , and <code>NewMovieFromHandle</code> (described on page 2-92, page 2-88, and page 2-90, respectively).
<code>trackID</code>	Specifies the ID value of the track for this operation.

DESCRIPTION

The `GetMovieTrack` function returns the track identifier that is appropriate to the specified track. If the function cannot locate the track, it sets this returned value to `nil`.

ERROR CODES

<code>invalidMovie</code>	-2010	This movie is corrupted or invalid
<code>trackIDNotFound</code>	-2029	Cannot locate a track with this ID value

SEE ALSO

You can obtain a track's ID value by calling the `GetTrackID` function, which is described in the next section. You can use a track's index value to obtain its track identifier by calling the `GetMovieIndTrack` function, which is described in the previous section.

GetTrackID

The `GetTrackID` function allows your application to determine a track's unique track ID value. This ID value remains unique throughout the life of the movie.

```
pascal long GetTrackID (Track theTrack);
```

`theTrack` Specifies the track for this operation. Your application obtains this track identifier from such Movie Toolbox functions as `NewMovieTrack` and `GetMovieTrack` (described on page 2-151 and page 2-204, respectively).

DESCRIPTION

The `GetTrackID` function returns the track's ID value. If the function could not determine the ID value, it sets this returned value to 0.

ERROR CODES

<code>invalidTrack</code>	<code>-2009</code>	This track is corrupted or invalid
---------------------------	--------------------	------------------------------------

GetTrackMovie

The `GetTrackMovie` function allows you to determine the movie that contains a specified track.

```
pascal Movie GetTrackMovie (Track theTrack);
```

`theTrack` Specifies the track for this operation. Your application obtains this track identifier from such Movie Toolbox functions as `NewMovieTrack` and `GetMovieTrack` (described on page 2-151 and page 2-204, respectively).

DESCRIPTION

The `GetTrackMovie` function returns the movie identifier that corresponds to the movie that contains the track. If the function could not locate the movie, it sets this returned value to `nil`.

ERROR CODES

<code>invalidTrack</code>	<code>-2009</code>	This track is corrupted or invalid
---------------------------	--------------------	------------------------------------