

## Summary of the Movie Toolbox

---

### C Summary

---

#### Constants

---

```
#define kFix1=      (0x00010000); /* fixed point value equal to 1.0 */

#define gestaltQuickTime 'qtim'      /* Movie Toolbox availability */
#define MovieFileType 'MooV'         /* movie file type */
#define VideoMediaType 'vide'        /* video media type */
#define SoundMediaType 'soun'        /* sound media type */
#define MediaHandlerType 'mhlr'      /* media handler type */
#define DataHandlerType 'dhlr'       /* data handler type */
#define TextMediaType 'text'         /* text media type */
#define GenericMediaType 'gnrc'      /* base media handler type */

#define DoTheRightThing = 0L          /* indicates default flag settings
                                       for Movie Toolbox functions */

/* sound volume values in trackVolume parameter of NewMovieTrack function */
#define kFullVolume  = 0x100          /* full, natural volume
                                       (8.8 format) */
#define kNoVolume    = 0              /* no volume */

/*
   constants for whichMediaTypes parameter of GetMovieNextInterestingTime
   function
*/

#define VisualMediaCharacteristic 'eyes' /* visual media */
#define AudioMediaCharacteristic 'ears' /* audio media */

enum
{
/* media quality settings in quality parameter of SetMediaQuality function */
    mediaQualityDraft      = 0x0000, /* lowest quality level */
    mediaQualityNormal     = 0x0040, /* acceptable quality level */
}
```

## Movie Toolbox

```

mediaQualityBetter    = 0x0080,      /* better quality level */
mediaQualityBest      = 0x00C0      /* best quality level */
};

enum
{
/*
values for callBackFlags field of QuickTime callback header structure used
by clock components to communicate scheduling information about a
callback event to the Movie Toolbox
*/
qtcBNeedsRateChanges    = 1,  /* rate changes */
qtcBNeedsTimeChanges    = 2,  /* time changes */
qtcBNeedsStartStopChanges = 4  /* time base changes at start &
                                stop times */
};

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

enum
{
movieInDataForkResID = -1  /* magic resource ID */
};

enum
{
/* flags for LoadIntoRAM functions */
keepInRam            = 1<<0,      /* load and make so data cannot be
                                purged */
unkeepInRam          = 1<<1,      /* mark data so it can be purged */
flushFromRam         = 1<<2,      /* empty handles and purge data from
                                memory */
};

```

## Movie Toolbox

```

loadForwardTrackEdits    = 1<<3,      /* load only data around
                                     track edits--play movie forward */
loadBackwardTrackEdits   = 1<<4      /* load only data around edits--
                                     play movie in reverse */
};

enum
{
/* flag for PasteHandleIntoMovie function */
pasteInParallel = 1 /* changes function to take contents and type of
                     handle and add to movie */
};

/* text description display flags used in AddTextSample and AddTESample */
enum
{
    dfDontDisplay          = 1<<0,      /* don't display the text */
    dfDontAutoScale        = 1<<1,      /* don't scale text as track bounds grows
                                     or shrinks */
    dfClipToTextBox        = 1<<2,      /* clip update to the text box */
    dfUseMovieBGColor       = 1<<3,      /* set text background to movie's
                                     background color */
    dfShrinkTextBoxToFit   = 1<<4,      /* compute minimum box to fit the
                                     sample */
    dfScrollIn              = 1<<5,      /* scroll text in until last of text is
                                     in view */
    dfScrollOut             = 1<<6      /* scroll text out until last of text is
                                     gone (if dfScrollIn is also set,
                                     scroll in then out */
    dfHorizScroll           = 1<<7,      /* scroll text horizontally--otherwise,
                                     it's vertical */
    dfReverseScroll         = 1<<8      /* vertically scroll down and
                                     horizontally scroll
                                     up--justification-dependent */
};

/* find flags for FindNextText function */
findTextEdgeOK           = 1<<0,      /* OK to find text at specified
                                     sample time */
findTextCaseSensitive     = 1<<1,      /* case-sensitive search */
findTextReverseSearch     = 1<<2,      /* search from sampleTime backward */
findTextWraparound       = 1<<3,      /* wrap search when beginning or end
                                     of movie is reached */

```

## Movie Toolbox

```

/* return display flags for application-defined text function */
enum
{
    txtProcDefaultDisplay    = 0,      /* use the media's default settings */
    txtProcDontDisplay       = 1,      /* don't display the text */
    txtProcDoDisplay         = 2       /* display the text */
};

enum
{
    hintsScrubMode           = 1<<0,   /* toolbox can display key frames when
                                         movie is repositioned */
    hintsAllowInterlace      = 1<<6,   /* use interlace option for compressor
                                         components */
    hintsUseSoundInterp      = 1<<7    /* turn on sound interpolation */
};
typedef unsigned long playHintsEnum;

```

## Data Types

---

```

typedef MovieRecord *Movie;           /* movie identifier */
typedef TrackRecord *Track;           /* track identifier */
typedef MediaRecord *Media;          /* media identifier */
typedef UserDataRecord *UserData;    /* user data list identifier */
typedef TrackEditStateRecord *TrackEditState;
                                     /* track edit state identifier */
typedef MovieEditStateRecord *MovieEditState;
                                     /* movie edit state identifier */
typedef long TimeValue;               /* time value field in time structure */
typedef long TimeScale;              /* time scale field in time structure */
typedef TimeBaseRecord *TimeBase;    /* time base identifier */
typedef CallbackRecord *QTCallback;  /* callback identifier */

typedef Int64 CompTimeValue;

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

```

## Movie Toolbox

```

struct TimeRecord
{
    CompTimeValue  value; /* time value as duration or absolute */
    TimeScale      scale; /* time scale as unit of time & number of units */
    TimeBase       base; /* reference to the time base */
};

typedef struct TimeRecord TimeRecord;

/* All sample descriptions start with this header. */

struct SampleDescription
{
    long    descSize;          /* total size in bytes of this structure */
    long    dataFormat;        /* format of the sample data */
    long    resvd1;            /* reserved--set to 0 */
    short   resvd2;            /* reserved--set to 0 */
    short   dataRefIndex;      /* reserved--set to 1 */
};

typedef struct SampleDescription SampleDescription;
typedef SampleDescription *SampleDescriptionPtr, **SampleDescriptionHandle;

struct SoundDescription
{
    long    descSize; /* total size in bytes of 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; /* which version is this data? */
    short   revlevel; /* which version of that codec did this? */
    long    vendor; /* which codec compressed this data? */
    short   numChannels; /* number of sound channels used by sample */
    short   sampleSize; /* number of bits in each sound sample */
    short   compressionID;
                /* sound compression used--set to 0 if none */
    short   packetSize; /* packet size for compression--set to 0 if
                        no compression */
    Fixed   sampleRate; /* rate at which sound samples were obtained */
};

typedef struct SoundDescription SoundDescription;
typedef SoundDescription *SoundDescriptionPtr, **SoundDescriptionHandle;

```

## Movie Toolbox

```

typedef struct TextDescription
{
    long    size;           /* total size of this text description
                           structure */
    long    type;           /* type of data in this structure ('text') */
    long    resvd1;         /* reserved for use by Apple--set to 0 */
    short   resvd2;         /* reserved for use by Apple--set to 0 */
    short   dataRefIndex;   /* index to data references */
    long    displayFlags;   /* display flags for text */
    long    textJustification;
                           /* text justification flags */
    RGBColor bgColor;       /* background color */
    Rect     defaultTextBox; /* location of the text within track bounds */
    ScrpSTElement defaultStyle;
                           /* default style (TextEdit structure) */
} TextDescription, *TextDescriptionPtr, **TextDescriptionHandle;

typedef struct TextDescription TextDescription;
typedef TextDescription *TextDescriptionPtr, **TextDescriptionHandle;

/* pointer to application-defined movie progress function */
typedef pascal OSErr (*MovieProgressProcPtr) (Movie theMovie, short message,
                                              short whatOperation, Fixed percentDone, long refcon);

/* pointer to application-defined cover function */
typedef pascal OSErr (*MovieRgnCoverProc) (Movie theMovie,
                                           RgnHandle changedRgn, long refcon);

typedef Handle MediaInformationHandle; /* data returned by media handler */
typedef ComponentInstance MediaHandler; /* media handler */
typedef Component MediaHandlerComponent; /* media handler component */
typedef ComponentInstance DataHandler; /* data handler */
typedef Component DataHandlerComponent; /* data handler component */
typedef ComponentResult HandlerError; /* error handler */
typedef ComponentInstance MovieController;
                           /* movie controller */

/* pointer to application-defined error-notification function */
typedef pascal void (*ErrorProcPtr) (OSErr theErr, long refcon);

/* pointer to application-defined movie preview callout function */
typedef pascal Boolean (*MoviePreviewCallOutProc) (long refcon);

```

## Movie Toolbox

```

enum
{
/* control flags for timeBaseFlags parameter of SetTimeBaseFlags function */
    loopTimeBase          = 1, /* whether time base loops */
    palindromeLoopTimeBase = 2 /* whether time base loops in palindrome
                                fashion */
};

typedef unsigned long TimeBaseFlags; /* control flags for time base */

/* pointer to application-defined callback function */
typedef pascal void (*QTCallbackProc)(QTCallback cb, long refcon);

struct QTCallbackHeader
{
    long    callBackFlags; /* clock component scheduling data flags */
    long    reserved1;     /* reserved for use by Apple */
    char    qtPrivate[40]; /* reserved for use by Apple */
};

struct MatrixRecord
{
    Fixed matrix[3][3];
};

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

typedef struct MatrixRecord MatrixRecord;
typedef MatrixRecord *MatrixRecordPtr; /* pointer to matrix structure */

struct FixedRect
{
    Fixed left; /* x coordinate of rectangle's upper-left corner */
    Fixed top; /* y coordinate of rectangle's upper-left corner */
    Fixed right; /* x coordinate of rectangle's lower-right corner */
    Fixed bottom; /* y coordinate of rectangle's lower-right corner */
};

typedef struct FixedRect FixedRect;

```

## Movie Toolbox

```

enum
{
/* progress function messages */
    movieProgressOpen          = 0, /* indicates start of a long operation */
    movieProgressUpdatePercent = 1, /* passes completion data to function */
    movieProgressClose         = 2  /* indicates end of a long operation */
};
typedef unsigned char movieProgressMessages;

enum
{
/*
    progress function operations that tell which function your application
    has called
*/
    progressOpFlatten          = 1, /* FlattenMovie or
                                     FlattenMovieData */
    progressOpInsertTrackSegment = 2, /* InsertTrackSegment */
    progressOpInsertMovieSegment = 3, /* InsertMovieSegment */
    progressOpPaste             = 4, /* PasteMovieSelection */
    progressOpAddMovieSelection = 5, /* AddMovieSelection */
    progressOpCopy              = 6, /* CopyMovieSelection */
    progressOpCut               = 7, /* CutMovieSelection */
    progressOpLoadMovieIntoRam   = 8, /* LoadMovieIntoRam */
    progressOpLoadTrackIntoRam   = 9, /* LoadTrackIntoRam */
    progressOpLoadMediaIntoRam   = 10, /* LoadMediaIntoRam */
    progressOpImportMovie        = 11, /* ConvertFileToMovieFile */
    progressOpExportMovie        = 12 /* ConvertMovieFile */
};
typedef unsigned char movieProgressOperations;

enum
{
/* NewMovie function flags */
    newMovieActive          = 1<<0, /* new movie is or
                                     is not active */
    newMovieDontResolveDataRefs = 1<<1, /* data reference
                                     resolution level */
    newMovieDontAskUnresolvedDataRefs = 1<<2, /* is user asked to locate
                                     files? */
};

```



## Movie Toolbox

```

    newMovieDontAutoAlternates      = 1<<3      /* are enabled tracks
                                                    selected from
                                                    alternate groups? */
};
typedef unsigned char newMovieFlagsEnum;

/* track usage flags in SetTrackUsage function */
enum
{
    trackUsageInMovie      = 1<<1,  /* track is used in movie */
    trackUsageInPreview    = 1<<2,  /* track is used in preview */
    trackUsageInPoster     = 1<<3   /* track is used in poster */
};
typedef unsigned char trackUsageEnum;

/* media sample flags in AddMediaSample function */
enum
{
    mediaSampleNotSync      = 1<<0,  /* sample to be added is not a
                                     sync sample */
    mediaSampleShadowSync   = 1<<1   /* sample is a shadow sync sample */
};
typedef unsigned char mediaSampleFlagsEnum;

enum
{
    /*
    interesting times flags in interestingTimeFlags parameter of
    GetMovieNextInterestingTime function
    */
    nextTimeMediaSample     = 1<<0,  /* finds next sample in movie's media */
    nextTimeMediaEdit       = 1<<1,  /* finds next sample group in movie's
                                     media */
    nextTimeTrackEdit       = 1<<2,  /* finds sample for next entry in edit
                                     list */
    nextTimeSyncSample      = 1<<3,  /* finds next sync sample in movie's
                                     media */
    nextTimeEdgeOK          = 1<<14,
                                /* to receive element data at specified
                                time */

```

## Movie Toolbox

```

nextTimeIgnoreActiveSegment
    = 1<<15
    /* look outside active segment for
       samples */
};
typedef unsigned short nextTimeFlagsEnum;

enum
{
/* movie-creation flags from CreateMovieFile function */
    createMovieFileDeleteCurFile    = 1L<<31, /* delete existing file? */
    createMovieFileDontCreateMovie   = 1L<<30, /* is new movie created? */
    createMovieFileDontOpenFile      = 1L<<29  /* is new movie file
                                                opened? */
};
typedef unsigned long createMovieFileFlagsEnum;

/* movie-flattening flags from FlattenFlags function */
enum
{
    flattenAddMovieToDataFork        = 1L<<0, /* movie placed in data fork */
    flattenActiveTracksOnly          = 1L<<2, /* enabled movie tracks added */
    flattenDontInterleaveFlatten     = 1L<<3 /* disable data storage
                                                optimizations */
};
typedef unsigned long movieFlattenFlagsEnum;

enum
{
/* movie scrap flags from PutMovieOnScrap function */
    movieScrapDontZeroScrap= 1<<0, /* is scrap cleared before movie is
                                     put on scrap? */
    movieScrapOnlyPutMovie = 1<<1 /* are other items placed on scrap along
                                     with movie? */
};
typedef unsigned char movieScrapFlagsEnum;

```

## Movie Toolbox

```

enum
{
/*
    callback flags from CallMeWhen function specify when callback
    should be called
*/
    triggerTimeFwd      = 0x0001, /* only when time is at positive rate */
    triggerTimeBwd      = 0x0002, /* only when time is at negative rate */
    triggerTimeEither    = 0x0003, /* at specified time without regard
                                   to rate */
    triggerRateLT        = 0x0004, /* whenever rate changes */
    triggerRateGT        = 0x0008, /* when changed rate greater than param2 */
    triggerRateEqual     = 0x0010, /* when changed rate equal to param2 */
    triggerRateLTE       = triggerRateLT | triggerRateEqual,
                        /* when rate less than or equal to
                           param2 */
    triggerRateGTE       = triggerRateGT | triggerRateEqual,
                        /* when rate greater than or equal to
                           param2 */
    triggerRateNotEqual  = triggerRateGT | triggerRateEqual | triggerRateLT,
                        /* when rate is not equal to param2 */
    triggerRateChange    = 0      /* whenever rate changes */
    triggerAtStart       = 0x0001,
                        /* at startup time */
    triggerAtStop        = 0x0002 /* at stop time */
};
typedef unsigned short QTCallBackFlags;

enum
{
/*
    flags returned by GetTimeBaseStatus function specify where time value in
    time structure lies
*/
    timeBaseBeforeStartTime = 1, /* before start time of time base */
    timeBaseAfterStopTime   = 2  /* after stop time of time base */
};
typedef unsigned long TimeBaseStatus;

```

## Movie Toolbox

```

enum
{
/*
    values for cbType parameter of NewCallBack function specify when event
    can be invoked
*/
    callBackAtTime      = 1,          /* at a specified time */
    callBackAtRate      = 2,          /* time base rate at specified value */
    callBackAtTimeJump  = 3,          /* time value jumps unexpectedly */
    callBackAtExtremes  = 4           /* time value at start time, stop time,
                                     or either */
    callBackAtInterrupt = 0x8000     /* at interrupt time */
};
typedef unsigned short QTCallBackType;

enum
{
    identityMatrixType    = 0x00,    /* identity matrix */
    translateMatrixType   = 0x01,    /* translation operation */
    scaleTranslateMatrixType = 0x03, /* translation & scaling operation */
    linearMatrixType      = 0x04,    /* rotation, skew, or shear
                                     operation */
    linearTranslateMatrixType = 0x05, /* translation & rotation, skew,
                                     or shear operation */
    perspectiveMatrixType = 0x06     /* perspective (nonlinear)
                                     operation */
};
typedef unsigned short MatrixFlags;

enum
{
/*
    values for the dataRefAttributes parameter of the GetMediaDataRef
    function
*/
    dataRefSelfReference = 1<<0, /* is reference to movie resource's
                                   data file? */
    dataRefWasNotResolved = 1<<1 /* did Movie Toolbox resolve reference? */
};
typedef unsigned long dataRefAttributesFlags;

```

## Movie Toolbox

```
enum
{
/* flags for SetMoviePlayHints and SetMediaPlayHints functions */
    hintsScrubMode      = 1<<0, /* mask == && (if flags == scrub on,
                                flags != scrub off) */

    hintsAllowInterlace = 1<<6,
    hintsUseSoundInterp = 1<<7
} ;
typedef unsigned long playHintsEnum;

enum
{
    mediaHandlerFlagGenericClient = 1 /* component flag--should be set for
                                     all media handler components that
                                     make use of generic media
                                     handler */
};
typedef unsigned long mediaHandlerFlagsEnum;
```

## Functions for Getting and Playing Movies

---

### Initializing the Movie Toolbox

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

### Error Functions

```
pascal OSErr GetMoviesError
                                (void);
pascal OSErr GetMoviesStickyError
                                (void);
pascal void ClearMoviesStickyError
                                (void);
pascal void SetMoviesErrorProc
                                (ErrorProcPtr errProc, long refcon);
```

### Movie Functions

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

## Movie Toolbox

```

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

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

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

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

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

pascal OSErr CloseMovieFile
    (short resRefNum);

pascal OSErr DeleteMovieFile
    (const FSSpec *fileSpec);

```

**Saving Movies**

```

pascal Boolean HasMovieChanged
    (Movie theMovie);

pascal void ClearMovieChanged
    (Movie theMovie);

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

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

pascal OSErr RemoveMovieResource
    (short resRefNum, short resId);

```

## Movie Toolbox

```

pascal OSErr PutMovieIntoHandle
                                (Movie theMovie, Handle publicMovie);

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

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

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

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

```

**Controlling Movie Playback**

```

pascal void StartMovie          (Movie theMovie);
pascal void StopMovie           (Movie theMovie);
pascal void GoToBeginningOfMovie
                                (Movie theMovie);
pascal void GoToEndOfMovie      (Movie theMovie);

```

**Movie Posters and Movie Previews**

```

pascal void SetTrackUsage       (Track theTrack, long usage);
pascal long GetTrackUsage       (Track theTrack);
pascal void ShowMoviePoster
                                (Movie theMovie);
pascal void SetPosterBox        (Movie theMovie, const Rect *boxRect);
pascal void GetPosterBox        (Movie theMovie, Rect *boxRect);
pascal void SetMoviePosterTime
                                (Movie theMovie, TimeValue posterTime);
pascal TimeValue GetMoviePosterTime
                                (Movie theMovie);
pascal void PlayMoviePreview
                                (Movie theMovie,
                                MoviePreviewCallOutProc callOutProc,
                                long refcon);

```

```

pascal void SetMoviePreviewMode
    (Movie theMovie, Boolean usePreview);

pascal Boolean GetMoviePreviewMode
    (Movie theMovie);

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

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

```

### Movies and Your Event Loop

```

pascal void MoviesTask      (Movie theMovie, long maxMilliSecToUse);
pascal Boolean IsMovieDone  (Movie theMovie);
pascal OSErr UpdateMovie    (Movie theMovie);
pascal Boolean PtInMovie    (Movie theMovie, Point pt);
pascal Boolean PtInTrack    (Track theTrack, Point pt);
pascal ComponentResult GetMovieStatus
    (Movie theMovie, Track *firstProblemTrack);
pascal ComponentResult GetTrackStatus
    (Track theTrack);

```

### Preferred Movie Settings

```

pascal void SetMoviePreferredRate
    (Movie theMovie, Fixed rate);

pascal Fixed GetMoviePreferredRate
    (Movie theMovie);

pascal void SetMoviePreferredVolume
    (Movie theMovie, short volume);

pascal short GetMoviePreferredVolume
    (Movie theMovie);

```

### Enhancing Movie Playback Performance

```

pascal OSErr PrerollMovie    (Movie theMovie, TimeValue time, Fixed Rate);
pascal void SetMovieActiveSegment
    (Movie theMovie, TimeValue startTime,
     TimeValue duration);

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

```



## Movie Toolbox

```

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

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

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

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

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

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

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

```

**Disabling Movies and Tracks**

```

pascal void SetMovieActive (Movie theMovie, Boolean active);

pascal Boolean GetMovieActive
                (Movie theMovie);

pascal void SetTrackEnabled
                (Track theTrack, Boolean isEnabled);

pascal Boolean GetTrackEnabled
                (Track theTrack);

```

**Generating Pictures From Movies**

```

pascal PicHandle GetMoviePict
                (Movie theMovie, TimeValue time);

pascal PicHandle GetMoviePosterPict
                (Movie theMovie);

pascal PicHandle GetTrackPict
                (Track theTrack, TimeValue time);

```

**Creating Tracks and Media Structures**

```

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

pascal void DisposeMovieTrack
                (Track theTrack);

```

## Movie Toolbox

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

pascal void DisposeTrackMedia
                           (Media theMedia);
```

**Working With Progress and Cover Functions**

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

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

**Functions That Modify Movie Properties**

---

**Working With Movie Spatial Characteristics**

```
pascal void SetMovieGWorld (Movie theMovie, CGrafPtr port, GDHandle gdh);
pascal void GetMovieGWorld (Movie theMovie, CGrafPtr *port, GDHandle *gdh);
pascal void SetMovieBox (Movie theMovie, const Rect *boxRect);
pascal void GetMovieBox (Movie theMovie, Rect *boxRect);
pascal RgnHandle GetMovieDisplayBoundsRgn
                           (Movie theMovie);
pascal RgnHandle GetMovieSegmentDisplayBoundsRgn
                           (Movie theMovie, TimeValue time,
                           TimeValue duration);
pascal void SetMovieDisplayClipRgn
                           (Movie theMovie, RgnHandle theClip);
pascal RgnHandle GetMovieDisplayClipRgn
                           (Movie theMovie);
pascal RgnHandle GetTrackDisplayBoundsRgn
                           (Track theTrack);
pascal RgnHandle GetTrackSegmentDisplayBoundsRgn
                           (Track theTrack, TimeValue time, TimeValue
                           duration);
pascal void SetTrackLayer (Track theTrack, short layer);
pascal short GetTrackLayer (Track theTrack);
pascal void SetMovieMatrix (Movie theMovie, const MatrixRecord *matrix);
pascal void GetMovieMatrix (Movie theMovie, MatrixRecord *matrix);
pascal RgnHandle GetMovieBoundsRgn
                           (Movie theMovie);
```

## Movie Toolbox

```

pascal RgnHandle GetTrackMovieBoundsRgn
                                (Track theTrack);

pascal void SetMovieClipRgn
                                (Movie theMovie, RgnHandle theClip);

pascal RgnHandle GetMovieClipRgn
                                (Movie theMovie);

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

pascal RgnHandle GetTrackBoundsRgn
                                (Track theTrack);

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

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

pascal void SetTrackClipRgn
                                (Track theTrack, RgnHandle theClip);

pascal RgnHandle GetTrackClipRgn
                                (Track theTrack);

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

pascal void DisposeMatte     (PixMapHandle theMatte);

```

**Working With Sound Volume**

```

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

pascal void SetTrackVolume   (Track theTrack, short volume);
pascal short GetTrackVolume
                                (Track theTrack);

```

**Working With Movie Time**

```

pascal TimeValue GetMovieDuration
                                (Movie theMovie);

pascal void SetMovieTimeValue
                                (Movie theMovie, TimeValue newtime);

pascal void SetMovieTime     (Movie theMovie, const TimeRecord *newtime);
pascal TimeValue GetMovieTime
                                (Movie theMovie, TimeRecord *currentTime);

pascal void SetMovieRate     (Movie theMovie, Fixed rate);
pascal Fixed GetMovieRate    (Movie theMovie);

```

## Movie Toolbox

```

pascal void SetMovieTimeScale
    (Movie theMovie, TimeScale timeScale);
pascal TimeScale GetMovieTimeScale
    (Movie theMovie);
pascal TimeBase GetMovieTimeBase
    (Movie theMovie);

```

**Working With Track Time**

```

pascal TimeValue GetTrackDuration
    (Track theTrack);
pascal void SetTrackOffset (Track theTrack, TimeValue movieOffsetTime);
pascal TimeValue GetTrackOffset
    (Track theTrack);
pascal TimeValue TrackTimeToMediaTime
    (TimeValue value, Track theTrack);

```

**Working With Media Time**

```

pascal TimeValue GetMediaDuration
    (Media theMedia);
pascal void SetMediaTimeScale
    (Media theMedia, TimeScale timeScale);
pascal TimeScale GetMediaTimeScale
    (Media theMedia);

```

**Finding Interesting Times**

```

pascal void GetMovieNextInterestingTime
    (Movie theMovie, short interestingTimeFlags,
     short numMediaTypes, const OSType
     *whichMediaTypes, TimeValue time, Fixed rate,
     TimeValue *interestingTime,
     TimeValue *interestingDuration);
pascal void GetTrackNextInterestingTime
    (Track theTrack, short interestingTimeFlags,
     TimeValue time, Fixed rate,
     TimeValue *interestingTime,
     TimeValue *interestingDuration);
pascal void GetMediaNextInterestingTime
    (Media theMedia, short interestingTimeFlags,
     TimeValue time, Fixed rate,
     TimeValue *interestingTime,
     TimeValue *interestingDuration);

```

**Locating a Movie's Tracks and Media Structures**

```

pascal long GetMovieTrackCount
                                (Movie theMovie);

pascal Track GetMovieIndTrack
                                (Movie theMovie, long index);

pascal Track GetMovieTrack      (Movie theMovie, long trackID);
pascal long GetTrackID          (Track theTrack);
pascal Movie GetTrackMovie      (Track theTrack);
pascal Media GetTrackMedia      (Track theTrack);
pascal Track GetMediaTrack      (Media theMedia);

```

**Working With Alternate Tracks**

```

pascal void SetMovieLanguage (Movie theMovie, long language);
pascal void SelectMovieAlternates
                                (Movie theMovie);

pascal void SetAutoTrackAlternatesEnabled
                                (Movie theMovie, Boolean enable);

pascal void SetTrackAlternate
                                (Track theTrack, Track alternateT);

pascal Track GetTrackAlternate
                                (Track theTrack);

pascal void SetMediaLanguage
                                (Media theMedia, short language);

pascal short GetMediaLanguage
                                (Media theMedia);

pascal void SetMediaQuality
                                (Media theMedia, short quality);

pascal short GetMediaQuality
                                (Media theMedia);

```

**Working With Data References**

```

pascal OSErr AddMediaDataRef
                                (Media theMedia, short *index, Handle dataRef,
                                 OSType dataRefType);

pascal OSErr SetMediaDataRef
                                (Media theMedia, short index, Handle dataRef,
                                 OSType dataRefType);

```

```

pascal OSErr GetMediaDataRefCount
    (Media theMedia, short *count);

pascal OSErr GetMediaDataRef
    (Media theMedia, short index, Handle *dataRef,
     OSType *dataRefType, long *dataRefAttributes);

```

### Determining Movie Creation and Modification Time

```

pascal unsigned long GetMovieCreationTime
    (Movie theMovie);

pascal unsigned long GetMovieModificationTime
    (Movie theMovie);

pascal unsigned long GetTrackCreationTime
    (Track theTrack);

pascal unsigned long GetTrackModificationTime
    (Track theTrack);

pascal unsigned long GetMediaCreationTime
    (Media theMedia);

pascal unsigned long GetMediaModificationTime
    (Media theMedia);

```

### Working With Media Samples

```

pascal long GetMovieDataSize
    (Movie theMovie, TimeValue startTime,
     TimeValue duration);

pascal long GetTrackDataSize
    (Track theTrack, TimeValue startTime,
     TimeValue duration);

pascal long GetMediaDataSize
    (Media theMedia, TimeValue startTime,
     TimeValue duration);

pascal long GetMediaSampleCount
    (Media theMedia);

pascal long GetMediaSampleDescriptionCount
    (Media theMedia);

pascal void GetMediaSampleDescription
    (Media theMedia, long index,
     SampleDescriptionHandle descH);

pascal OSErr SetMediaSampleDescription
    (Media theMedia, long index,
     SampleDescriptionHandle descH);

```

## Movie Toolbox

```

pascal void MediaTimeToSampleNum
    (Media theMedia, TimeValue time,
     long *sampleNum, TimeValue *sampleTime,
     TimeValue *sampleDuration);

pascal void SampleNumToMediaTime
    (Media theMedia, long logicalSampleNum,
     TimeValue *sampleTime,
     TimeValue *sampleDuration);

```

**Working With Movie User Data**

```

pascal UserData GetMovieUserData
    (Movie theMovie);

pascal UserData GetTrackUserData
    (Track theTrack);

pascal UserData GetMediaUserData
    (Media theMedia);

pascal long GetNextUserDataType
    (UserData theUserData, OSType udType);

pascal short CountUserDataTypes
    (UserData theUserData, OSType udType);

pascal OSErr AddUserData
    (UserData theUserData, Handle data,
     OSType udType);

pascal OSErr GetUserData
    (UserData theUserData, Handle data,
     OSType udType, long index);

pascal OSErr RemoveUserData
    (UserData theUserData, OSType udType,
     long index);

pascal OSErr AddUserDataText
    (UserData theUserData, Handle data, OSType
     udType, long index, short itlRegionTag);

pascal OSErr GetUserDataText
    (UserData theUserData, Handle data,
     OSType udType, long index, short itlRegionTag);

pascal OSErr RemoveUserDataText
    (UserData theUserData, OSType udType,
     long index, short itlRegionTag);

pascal OSErr SetUserDataItem
    (UserData theUserData, void *data, long size,
     OSType udType, long index);

pascal OSErr GetUserDataItem
    (UserData theUserData, void *data, long size,
     OSType udType, long index);

```

## Movie Toolbox

```

pascal OSErr NewUserData      (UserData *theUserData);
pascal OSErr DisposeUserData
                                (UserData theUserData);
pascal OSErr PutUserDataIntoHandle
                                (UserData theUserData, Handle h);
pascal OSErr NewUserDataFromHandle
                                (Handle h, UserData *theUserData);

```

## Functions for Editing Movies

---

### Editing Movies

```

pascal OSErr PutMovieOnScrap
                                (Movie theMovie, long movieScrapFlags);
pascal Movie NewMovieFromScrap
                                (long newMovieFlags);
pascal void SetMovieSelection
                                (Movie theMovie, TimeValue selectionTime,
                                 TimeValue selectionDuration);
pascal void GetMovieSelection
                                (Movie theMovie, TimeValue *selectionTime,
                                 TimeValue *selectionDuration);
pascal Movie CutMovieSelection
                                (Movie theMovie);
pascal Movie CopyMovieSelection
                                (Movie theMovie);
pascal void PasteMovieSelection
                                (Movie theMovie, Movie src);
pascal void AddMovieSelection
                                (Movie theMovie, Movie src);
pascal void ClearMovieSelection
                                (Movie theMovie);
pascal Component IsScrapMovie
                                (Track targetTrack);
pascal OSErr PasteHandleIntoMovie
                                (Handle h, OSType handleType, Movie theMovie,
                                 long flags, ComponentInstance userComp);
pascal OSErr PutMovieIntoTypedHandle
                                (Movie theMovie, Track targetTrack,
                                 OSType handleType, Handle publicMovie,
                                 TimeValue start, TimeValue dur,
                                 long flags, ComponentInstance userComp);

```



**Undo for Movies**

```

pascal MovieEditState NewMovieEditState
    (Movie theMovie);

pascal OSErr UseMovieEditState
    (Movie theMovie, MovieEditState toState);

pascal OSErr DisposeMovieEditState
    (MovieEditState state);

```

**Low-Level Movie-Editing Functions**

```

pascal OSErr InsertMovieSegment
    (Movie srcMovie, Movie dstMovie,
     TimeValue srcIn,
     TimeValue srcDuration, TimeValue dstIn);

pascal OSErr InsertEmptyMovieSegment
    (Movie dstMovie, TimeValue dstIn,
     TimeValue dstDuration);

pascal OSErr DeleteMovieSegment
    (Movie theMovie, TimeValue in, TimeValue
     duration);

pascal OSErr ScaleMovieSegment
    (Movie theMovie, TimeValue in,
     TimeValue oldDuration, TimeValue newDuration);

pascal OSErr CopyMovieSettings
    (Movie srcMovie, Movie dstMovie);

```

**Editing Tracks**

```

pascal OSErr InsertTrackSegment
    (Track srcTrack, Track dstTrack,
     TimeValue srcIn, TimeValue srcDuration,
     TimeValue dstIn);

pascal OSErr InsertEmptyTrackSegment
    (Track dstTrack, TimeValue dstIn,
     TimeValue dstDuration);

pascal OSErr InsertMediaIntoTrack
    (Track theTrack, TimeValue trackStart,
     TimeValue mediaTime,
     TimeValue mediaDuration, Fixed mediaRate);

pascal OSErr DeleteTrackSegment
    (Track theTrack, TimeValue in,
     TimeValue duration);

```

```

pascal OSErr ScaleTrackSegment
    (Track theTrack, TimeValue in,
     TimeValue oldDuration, TimeValue newDuration);

pascal OSErr CopyTrackSettings
    (Track srcTrack, Track dstTrack);

pascal Fixed GetTrackEditRate
    (Track theTrack, TimeValue atTime);

```

### Undo for Tracks

```

pascal TrackEditState NewTrackEditState
    (Track theTrack);

pascal OSErr UseTrackEditState
    (Track theTrack, TrackEditState state);

pascal OSErr DisposeTrackEditState
    (TrackEditState state);

```

### Adding Samples to Media Structures

```

pascal OSErr BeginMediaEdits
    (Media theMedia);

pascal OSErr EndMediaEdits (Media theMedia);

pascal OSErr AddMediaSample (Media theMedia, Handle dataIn, long inOffset,
    unsigned long size,
    TimeValue durationPerSample,
    SampleDescriptionHandle sampleDescriptionH,
    long numberOfSamples, short sampleFlags,
    TimeValue *sampleTime);

pascal OSErr AddMediaSampleReference
    (Media theMedia, long dataOffset,
    unsigned long size,
    TimeValue durationPerSample,
    SampleDescriptionHandle sampleDescriptionH,
    long numberOfSamples, short sampleFlags,
    TimeValue *sampleTime);

pascal OSErr GetMediaSample
    (Media theMedia, Handle dataOut,
    long maxSizeToGrow, long *size, TimeValue time,
    TimeValue *sampleTime,
    TimeValue *durationPerSample,
    SampleDescriptionHandle sampleDescriptionH,
    long *sampleDescriptionIndex,
    long maxNumberOfSamples,
    long *numberOfSamples, short *sampleFlags);

```

```

pascal OSErr GetMediaSampleReference
    (Media theMedia, long *dataOffset, long *size,
     TimeValue time, TimeValue *sampleTime,
     TimeValue *durationPerSample,
     SampleDescriptionHandle sampleDescriptionH,
     long *sampleDescriptionIndex,
     long maxNumberOfSamples,
     long *numberOfSamples, short *sampleFlags);

```

## Media Functions

---

### Selecting Media Handlers

```

pascal void GetMediaHandlerDescription
    (Media theMedia, OSType *mediaType,
     Str255 creatorName,
     OSType *creatorManufacturer);

pascal MediaHandler GetMediaHandler
    (Media theMedia);

pascal OSErr SetMediaHandler
    (Media theMedia, MediaHandlerComponent mh);

pascal void GetMediaDataHandlerDescription
    (Media theMedia, short index, OSType *dhType,
     Str255 creatorName,
     OSType *creatorManufacturer);

pascal DataHandler GetMediaDataHandler
    (Media theMedia, short index);

pascal OSErr SetMediaDataHandler
    (Media theMedia, short index,
     DataHandlerComponent dataHandler);

```

### Video Media Handler Functions

```

pascal HandlerError SetVideoMediaGraphicsMode
    (MediaHandler mh, long graphicsMode,
     const RGBColor *opColor);

pascal HandlerError GetVideoMediaGraphicsMode
    (MediaHandler mh, long *graphicsMode,
     RGBColor *opColor);

```

**Sound Media Handler Functions**

```
pascal HandlerError SetSoundMediaBalance
    (MediaHandler mh, short balance);

pascal HandlerError GetSoundMediaBalance
    (MediaHandler mh, short *balance);
```

**Text Media Handler Functions**

```
pascal ComponentResult AddTextSample
    (MediaHandler mh, Ptr text,
     unsigned long size, short fontNum,
     short fontSize, Style textFace,
     RGBColor *textColor,
     RGBColor *backColor,
     short textJustification,
     Rect *textBox, long displayFlags,
     TimeValue scrollDelay,
     short hiliteStart, short hiliteEnd,
     RGBColor rgbHiliteColor,
     TimeValue duration, TimeValue *sampleTime);

pascal ComponentResult AddTESample
    (MediaHandler mh, TEHandle hTE,
     RGBColor *backColor, short textJustification,
     Rect *textBox, long displayFlags,
     TimeValue scrollDelay, short hiliteStart,
     short hiliteEnd, RGBColor rgbHiliteColor,
     TimeValue duration,
     TimeValue *sampleTime);

pascal ComponentResult AddHiliteSample
    (MediaHandler mh, short hiliteStart,
     short hiliteEnd,
     RGBColor *rgbHiliteColor, TimeValue duration,
     TimeValue *sampleTime);

pascal ComponentResult FindNextText
    (MediaHandler mh, Ptr Text, long size,
     short findFlags, TimeValue startTime,
     TimeValue *foundTime, TimeValue *foundDuration,
     long *offset);

pascal ComponentResult HiliteTextSample
    (MediaHandler mh, TimeValue sampleTime,
     short hiliteStart, short hiliteEnd,
     RGBColor *rgbHiliteColor);
```

```
pascal ComponentResult SetTextProc
    (MediaHandler mh, TextMediaProcPtr TextProc,
     long refcon);
```

## Functions for Creating File Previews

---

```
pascal OSErr MakeFilePreview
    (short resRefNum,
     ProgressProcRecordPtr progress);

pascal OSErr AddFilePreview
    (short resRefNum, OSType previewType,
     Handle previewData);
```

## Functions for Displaying File Previews

---

```
pascal void SFGetFilePreview
    (Point where, ConstStr255Param prompt,
     FileFilterProcPtr fileFilter, short numTypes,
     SFTypeList typeList, DlgHookProcPtr dlgHook,
     SFReply *reply);

pascal void SFPGetFilePreview
    (Point where, ConstStr255Param prompt,
     FileFilterProcPtr fileFilter, short numTypes,
     SFTypeList typeList, DlgHookProcPtr dlgHook,
     SFReply *reply, short dlgID,
     ModalFilterProcPtr filterProc);

pascal void StandardGetFilePreview
    (FileFilterProcPtr fileFilter, short numTypes,
     SFTypeList typeList, StandardFileReply *reply);

pascal void CustomGetFilePreview
    (FileFilterYDProcPtr fileFilter,
     short numTypes, SFTypeList typeList,
     StandardFileReply *reply, short dlgID,
     Point where, DlgHookYDProcPtr dlgHook,
     ModalFilterYDProcPtr filterProc,
     short *activeList,
     ActivateYDProcPtr activateProc,
     void *yourDataPtr);
```

## Time Base Functions

---

### Creating and Disposing of Time Bases

```
pascal TimeBase NewTimeBase
                                (void);

pascal void DisposeTimeBase
                                (TimeBase tb);

pascal void SetMovieMasterClock
                                (Movie theMovie, Component clockMeister,
                                 const TimeRecord *slaveZero);

pascal void SetMovieMasterTimeBase
                                (Movie theMovie, TimeBase tb,
                                 const TimeRecord *slaveZero);

pascal void SetTimeBaseMasterClock
                                (TimeBase slave, Component clockMeister,
                                 const TimeRecord *slaveZero);

pascal ComponentInstance GetTimeBaseMasterClock
                                (TimeBase tb);

pascal void SetTimeBaseMasterTimeBase
                                (TimeBase slave, TimeBase master,
                                 const TimeRecord *slaveZero);

pascal TimeBase GetTimeBaseMasterTimeBase
                                (TimeBase tb);

pascal void SetTimeBaseZero
                                (TimeBase tb, TimeRecord *zero);
```

### Working With Time Base Values

```
pascal void SetTimeBaseTime
                                (TimeBase tb, const TimeRecord *tr);

pascal void SetTimeBaseValue
                                (TimeBase tb, TimeValue t, TimeScale s);

pascal TimeValue GetTimeBaseTime
                                (TimeBase tb, TimeScale s, TimeRecord *tr);

pascal void SetTimeBaseRate
                                (TimeBase tb, Fixed r);

pascal Fixed GetTimeBaseRate
                                (TimeBase tb);
```

## Movie Toolbox

```

pascal Fixed GetTimeBaseEffectiveRate
                                (TimeBase tb);
pascal void SetTimeBaseStartTime
                                (TimeBase tb, const TimeRecord *tr);
pascal TimeValue GetTimeBaseStartTime
                                (TimeBase tb, TimeScale s, TimeRecord *tr);
pascal void SetTimeBaseStopTime
                                (TimeBase tb, const TimeRecord *tr);
pascal TimeValue GetTimeBaseStopTime
                                (TimeBase tb, TimeScale s, TimeRecord *tr);
pascal void SetTimeBaseFlags
                                (TimeBase tb, long timeBaseFlags);
pascal long GetTimeBaseFlags
                                (TimeBase tb);
pascal long GetTimeBaseStatus
                                (TimeBase tb, TimeRecord *unpinnedTime);

```

**Working With Times**

```

pascal void AddTime              (TimeRecord *dst, const TimeRecord *src);
pascal void SubtractTime         (TimeRecord *dst, const TimeRecord *src);
pascal void ConvertTime          (TimeRecord *inout, TimeBase newBase);
pascal void ConvertTimeScale     (TimeRecord *inout, TimeScale newScale);

```

**Time Base Callback Functions**

```

pascal QTCallback NewCallback
                                (TimeBase tb, short cbType);
pascal OSErr CallMeWhen         (QTCallback cb,
                                QTCallbackProc callBackProc, long refcon,
                                long param1, long param2, long param3);
pascal void CancelCallback      (QTCallback cb);
pascal void DisposeCallback
                                (QTCallback cb);
pascal TimeBase GetCallbackTimeBase
                                (QTCallback cb);
pascal short GetCallbackType
                                (QTCallback cb);

```

## Matrix Functions

---

```

pascal void SetIdentityMatrix
                                (MatrixRecord *matrix);

pascal short GetMatrixType      (MatrixRecordPtr m);

pascal void CopyMatrix          (MatrixRecordPtr m1, MatrixRecord *m2);

pascal Boolean EqualMatrix      (const MatrixRecord *m1,
                                const MatrixRecord *m2);

pascal void TranslateMatrix
                                (MatrixRecord *m, Fixed deltaH, Fixed deltaV);

pascal void ScaleMatrix         (MatrixRecord *m, Fixed scaleX,
                                Fixed scaleY, Fixed aboutX, Fixed aboutY);

pascal void RotateMatrix        (MatrixRecord *m, Fixed degrees,
                                Fixed aboutX, Fixed aboutY);

pascal void SkewMatrix          (MatrixRecord *m, Fixed skewX, Fixed skewY,
                                Fixed aboutX, Fixed aboutY);

pascal void ConcatMatrix        (MatrixRecord *a, MatrixRecord *b);

pascal Boolean InverseMatrix
                                (MatrixRecord *m, MatrixRecord *im);

pascal OSErr TransformPoints
                                (MatrixRecord *mp, Point *pt1, long count);

pascal OSErr TransformFixedPoints
                                (MatrixRecord *m, FixedPoint *fpt, long count);

pascal Boolean TransformRect
                                (MatrixRecord *m, Rect *r, FixedPoint *fpp);

pascal Boolean TransformFixedRect
                                (MatrixRecord *m, FixedRect *fr,
                                FixedPoint *fpp);

pascal OSErr TransformRgn       (MatrixRecordPtr mp, RgnHandle r);

pascal void RectMatrix          (MatrixRecord *matrix, Rect *srcRect,
                                Rect *dstRect);

pascal void MapMatrix           (MatrixRecord *matrix, Rect *fromRect,
                                Rect *toRect);

```



## Application-Defined Functions

---

### Progress Functions

```
pascal OSErr MyProgressProc
                                (Movie theMovie, short message,
                                 short whatOperation,
                                 Fixed percentDone, long refcon);
```

### Cover Functions

```
pascal OSErr MyCoverProc      (Movie theMovie, RgnHandle changedRgn,
                                long refcon);
```

### Error-Notification Functions

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

### Movie Callout Functions

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

### File Filter Functions

```
pascal Boolean MyFileFilter
                                (ParmBlkPtr parmBlock);
```

### Custom Dialog Functions

```
pascal short MyDlgHook        (short item, DialogPtr theDialog,
                                Ptr myDataPtr);
```

### Modal-Dialog Filter Functions

```
pascal Boolean MyModalFilter
                                (DialogPtr theDialog, EventRecord *theEvent,
                                 short itemHit, Ptr myDataPtr);
```

### Standard File Activation Functions

```
pascal void MyActivateProc    (DialogPtr theDialog, short itemNo, Boolean
                                activating, Ptr myDataPtr);
```

### Callback Event Functions

```
pascal void MyCallBackProc    (QTCallBack cb, long refcon);
```

**Text Functions**

```
pascal OSErr MyTextProc      (Handle theText, Movie theMovie,
                             short *displayFlag, long refcon);
```

**Pascal Summary**

---

**Constants**

---

```
CONST
    kFix1                      = $00010000; {fixed point value equal }
                                { to 1.0}
    gestaltQuickTime          = 'qtim';    {Movie Toolbox availability}

    MovieFileType              = 'MooV';    {movie file type}

    VideoMediaType             = 'vide';    {video media type}
    SoundMediaType             = 'soun';    {sound media type}
    MediaHandlerType           = 'mhlr';    {media handler type}
    DataHandlerType            = 'dhlr';    {data handler type}
    TextMediaType              = 'text';    {text media type}
    GenericMediaType            = 'gnrc';    {base media handler type}

    DoTheRightThing = 0L                      {indicates default flag }
                                              { setting for Movie }
                                              { Toolbox functions}

    {progress procedure messages}
    movieProgressOpen           = 0;          {start of a long operation}
    movieProgressUpdatePercent  = 1;          {completion data to }
                                              { procedure}
    movieProgressClose          = 2;          {end of a long operation}

    {progress procedure operations that indicate which routine }
    { your application has called}
    progressOpFlatten           = 1;          {FlattenMovie or }
                                              { FlattenMovieData}
    progressOpInsertTrackSegment = 2;          {InsertTrackSegment}
    progressOpInsertMovieSegment = 3;          {InsertMovieSegment}
    progressOpPaste              = 4;          {PasteMovieSelection}
    progressOpAddMovieSelection  = 5;          {AddMovieSelection}
    progressOpCopy               = 6;          {CopyMovieSelection}
```

## Movie Toolbox

```

progressOpCut                = 7;          {CutMovieSelection}
progressOpLoadMovieIntoRam    = 8;          {LoadMovieIntoRam}
progressOpLoadTrackIntoRam    = 9;          {LoadTrackIntoRam}
progressOpLoadMediaIntoRam    = 10;         {LoadMediaIntoRam}
progressOpImportMovie         = 11;         {ConvertFileToMovieFile}
progressOpExportMovie         = 12;         {ConvertMovieFile}

{NewMovie function flags}
newMovieActive                = $1;         {is new movie active?}
newMovieDontResolveDataRefs    = $2;         {how data references are }
                                         { resolved in movie resource}
newMovieDontAskUnresolvedDataRefs = $4;     {is user asked to locate }
                                         { files? }
newMovieDontAutoAlternate      = $8;         {are enabled tracks }
                                         { selected from alternate }
                                         { groups?}

{sound volume values in trackVolume parameter of NewMovieTrack }
{ function}
kFullVolume                   = $100; {full, natural volume }
                                         { 8.8 format}
kNoVolume                     = 0;         {sets track to no volume}

{constants for whichMediaTypes parameter of }
{ GetMovieNextInterestingTime function}
VisualMediaCharacteristic 'eyes'      {visual media type}
AudioMediaCharacteristic 'ears'       {audio media type}

{track usage flags in SetTrackUsage procedure}
trackUsageInMovie              = $2;        {track is used in movie}
trackUsageInPreview            = $4;        {track is used in preview}
trackUsageInPoster             = $8;        {track is used in poster}

{media sample flags in AddMediaSample function}
mediaSampleNotSync             = 1;         {sample to be added not a }
                                         { sync sample}
mediaSampleShadowSync          = 2;         {sample is a shadow }
                                         { sync sample}

{media quality settings in quality parameter of }
{ SetMediaQuality procedure}
mediaQualityDraft              = $0000; {lowest quality level}

```

## Movie Toolbox

```

mediaQualityNormal          = $0040; {acceptable quality level}
mediaQualityBetter          = $0080; {better quality level}
mediaQualityBest            = $00C0; {best quality level}

{interesting times flags in interestingTimeFlags parameter }
{ of GetMovieNextInterestingTime procedure specify searching criteria}
nextTimeMediaSample         = $1;      {next sample in movie's }
                                   { media}
nextTimeMediaEdit           = $2;      {next sample group in media}
nextTimeTrackEdit           = $4;      {sample for next entry }
                                   { in edit list}
nextTimeSyncSample          = $8;      {next sync sample in }
                                   { movie's media}
nextTimeEdgeOK              = $2000;   {get specified time }
                                   { element data}
nextTimeIgnoreActiveSegment = $4000;   {outside active segment}

{flag for resID parameter of NewMovieFile function}
movieInDataForkResID        = -1;      {magic resource ID}

{movie-creation flags from CreateMovieFile function}
createMovieFileDeleteCurFile = $80000000;{delete existing file?}
createMovieFileDontCreateMovie = $40000000;{new movie created?}
createMovieFileDontOpenFile   = $20000000;{new movie file opened?}

{movie-flattening flags from FlattenFlags procedure}
flattenAddMovieToDataFork     = $1;      {movie in data fork of }
                                   { new movie file}
flattenActiveTracksOnly       = $4;      {enabled tracks added }
                                   { to movie file}
flattenDontInterleaveFlatten = $8;      {disables data }
                                   { storage optimizations}

{movie scrap flags from PutMovieOnScrap function}
movieScrapDontZeroScrap       = $1;      {is scrap cleared before }
                                   { movie on scrap?}
movieScrapOnlyPutMovie        = $2;      {are other items on }
                                   { scrap with movie?}

mediaHandlerFlagGenericClient = 1; {component flag--should be set }
                                   { for all media handlers }
                                   { components that use generic }
                                   { media handlers}

```

## Movie Toolbox

```

{callback flags from CallMeWhen function specify when callback }
{ should be called}
triggerTimeFwd           = $0001;    {time is at positive rate}
triggerTimeBwd           = $0002;    {time is at negative rate}
triggerTimeEither        = $0003;    {without regard to rate}
triggerRateLT            = $0004;    {whenever rate changes}
triggerRateGT            = $0008;    {rate change less than }
                                { param2}
triggerRateEqual         = $0010;    {rate change equal to }
                                { param2}
triggerRateLTE           = $0014;    {rate change less than or }
                                { equal to param2}
triggerRateGTE           = $0018;    {rate change greater than }
                                { or equal to param2 specification}
triggerRateNotEqual      = $001C;    {rate change not equal to param2}
triggerRateChange        = 0;        {whenever rate changes}
triggerAtStart           = $0001;    {at start time}
triggerAtStop            = $0002;    {at stop time}

{flags returned by GetTimeBaseStatus function specify where }
{ time value in time record lies}
timeBaseBeforeStartTime  = 1;        {before start time of time base}
timeBaseAfterStopTime    = 2;        {after stop time of time base}

{values for cbType parameter of NewCallBack function specify when }
{ event can be invoked}
callBackAtTime           = 1;        {at a specified time}
callBackAtRate           = 2;        {rate for time base reaches value}
callBackAtTimeJump       = 3;        {when time value changes }
                                { by amount differing from }
                                { its rate}
callBackAtExtremes       = 4;        {at start time, at stop time, }
                                { or both}
callBackAtInterrupt      = $8000;    {at interrupt time}

{values for callBackFlags field of QuickTime callback header record }
{ used by clock components to communicate scheduling information }
{ about a callback event to the Movie Toolbox}
qtcBNeedsRateChanges     = 1;        {rate changes}
qtcBNeedsTimeChanges     = 2;        {time changes}
qtcBNeedsStartStopChanges = 4;;      {changes in time base start/stop}

```

## Movie Toolbox

```

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

{control flags for timeBaseFlags parameter of SetTimeBaseFlags }
{ function}
loopTimeBase                = 1;      {whether time base loops}
palindromeLoopTimeBase      = 2;      {whether time base loops }
                                   { in palindrome fashion}

{flags for LoadIntoRAM functions}
keepInRam                   = 1;      {load and make so data cannot be }
                                   { purged}
unkeepInRam                 = 2;      {mark data so it can be purged}
flushFromRam                = 4;      {empty handles and purge data }
                                   { from memory}
loadForwardTrackEdits       = 8;      {load only data around track }
                                   { edits--play movie forward}
loadBackwardTrackEdits      = 16;     {load only data around track}
                                   { edits--play movie in reverse}

{flag for PasteHandleIntoMovie function}
pasteInParallel             = 1;      {changes function to take }
                                   { contents and type of handle }
                                   { and add to movie}

{text description display flags used in AddTextSample and }
{ AddTESample functions}
dfDontDisplay               = 1;      {don't display the text}
dfDontAutoScale             = 2;      {don't scale text as track }
                                   { boundaries grow or shrink}
dfClipToTextBox             = 4;      {clip update to the text box}
dfUseMovieBGColor           = 8;      {set text background to }
                                   { movie's background color}
dfShrinkTextBoxToFit        = 16;     {compute minimum box to fit }
                                   { the sample}
dfScrollIn                  = 32;     {scroll text in until last }
                                   { of text is in view}
dfScrollOut                 = 64;     {scroll text out until last }
                                   { of text is gone}

```

## Movie Toolbox

```

dfHorizScroll      = 128;    {scroll text horizontally}
dfReverseScroll    = 256;    {vertical text scrolls down, }
                                { horizontal text scrolls }
                                { backward; justification dependent}

{values returned by the GetMatrixType function}
identityMatrixType = $00;    {matrix is identity}
translateMatrixType = $01;    {matrix translates}
scaleMatrixType    = $02;    {matrix scales}
scaleTranslateMatrixType = $03; {matrix scales and translates}
linearMatrixType    = $04;    {matrix is general 2 x 2}
linearTranslateMatrixType = $05; {matrix is general 2 x 2 }
                                { and translates}
perspectiveMatrixType = $06; {matrix is general 3 x 3}

{return display flags for application-defined text function}
txtProcDefaultDisplay = 0;    {use the media's default settings}
txtProcDontDisplay    = 1;    {don't display the text}
txtProcDoDisplay      = 2;    {do display the text}

{find flags for FindNextTextFunction}
findTextEdgeOK        = 1;    {OK to find text at specified }
                                { sample time}
findTextCaseSensitive = 2;    {case-sensitive search}
findTextReverseSearch = 4;    {search from sample time backward}
findTextWrapAround    = 8;    {wrap search when beginning or }
                                { end of movie is reached}

{hints constants for play hints functions}
hintsScrubMode        = $1;    {toolbox can display key frames }
                                { when movie is repositioned}
hintsAllowInterlace   = $40;   {use interlace option for }
                                { compressor components}
hintsUseSoundInterp   = $80;   {turns on sound interpolation}

```

## Data Types

---

```

TYPE  Movie      = ^MovieRecord;    {movie identifier}
      Track      = ^TrackRecord;    {track identifier}
      Media      = ^MediaRecord;    {media identifier}
      UserData   = ^UserDataRecord;  {user data list identifier}
      TrackEditState = ^TrackEditStateRecord; {track edit state identifier}
      MovieEditState = ^MovieEditStateRecord; {movie edit state identifier}
      TimeValue   = LongInt;         {time value}

```

## Movie Toolbox

```

TimeScale      = LongInt;           {time scale in time record}
TimeBase       = ^TimeBaseRecord;  {time base identifier}
TimeBaseStatus = LongInt;           {time base statistics}
QTCallBack     = ^CallBackRecord;  {callback identifier}
QTCallBackProc = procPtr;           {callback function }
                                         { identifier}

Int64          = CompTimeValue;    {time value in time }
                                         { record}

playHintsEnum  = LongInt;           {play hints enumeration}
movieFlattenFlagsEnum = LongInt;    {movie flatteners flags }
                                         { enumeration}

createMovieFileFlagsEnum = LongInt; {movie creation flags}
nextTimeFlagsEnum      = Byte;      {next time flags}

Int64 =
RECORD
    hi:      LongInt; {high-order bits of value field in time record}
    lo:      LongInt; {low-order bits of value field in time record}
END;

TimeRecord =
RECORD
    value:      CompTimeValue; {time value as duration or }
                                         { absolute time}
    scale:      TimeScale;      {time scale as time units}
    base:      TimeBase;        {reference to the time base}
END;

SampleDescriptionPtr= ^SampleDescription;{ptr to sample description}
SampleDescriptionHandle = ^SampleDescriptionPtr;{handle to sample }
                                         { description record}

SampleDescription =
RECORD
    descSize:      LongInt;      {total size in bytes of this record}
    dataFormat:    LongInt;      {format of the sample data}
    resvd1:        LongInt;      {reserved--set to 0}
    resvd2:        Integer;      {reserved--set to 0}
    dataRefIndex:  Integer;      {reserved--set to 1}

END;

SoundDescriptionPtr      = ^SoundDescription; {ptr to sound description}
SoundDescriptionHandle   = ^SoundDescriptionPtr; {handle to sound }
                                         { description record}

```



## Movie Toolbox

```

SoundDescription =
RECORD
    descSize:      LongInt; {total size in bytes of this record}
    dataFormat:    LongInt; {format of the sound data}
    resvd1:        LongInt; {reserved--set to 0}
    resvd2:        Integer; {reserved--set to 0}
    dataRefIndex:  Integer; {reserved--set to 1}
    version:       Integer; {which version is this data? }
                    { (set to 0)}
    revlevel:      Integer; {which version of the compressor }
                    { component did this? (set to 0)}
    vendor:        LongInt; {whose compressor component }
                    { compressed this data? (set to 0)}
    numChannels:   Integer; {number of sound channels}
    sampleSize:    Integer; {number of bits in each sample;}
    compressionID: Integer; {sound compression used--0 if none}
    packetSize:    Integer; {packet size for compression--0 if }
                    { no compression}
    sampleRate:    Fixed;   {rate at which sound samples }
                    { were obtained}

END;

```

```

TextDescriptionPtr = ^TextDescription;
TextDescriptionHandle = ^TextDescriptionPtr;
TextDescription =
RECORD
    descSize:      LongInt;      {total size of this text }
                                { description record}
    dataFormat:    LongInt;      {type of data in this record }
                                { ('text')}
    resvd1:        LongInt;      {reserved for use by Apple-- }
                                { set to 0}
    resvd2:        Integer;      {reserved for use by Apple-- }
                                { set to 0}
    dataRefIndex:  Integer;      {index to data references}
    displayFlags:  LongInt;      {display flags for text}
    textJustification: LongInt;  {text justification flags}
    bgColor:       RGBColor;     {background color}
    defaultTextBox: Rect;        {location of the text within }
                                { track bounds}
    defaultStyle:  ScrpSTElement; {default style (TextEdit }
                                { record)}

END;

```

## Movie Toolbox

```

MovieProgressProcPtr    = ProcPtr;    {pointer to application-defined }
                                { movie progress procedure}

MovieRgnCoverProc       = ProcPtr;    {a pointer to application-defined }
                                { cover procedure}

MediaInformationHandle   = Handle;      {data returned }
                                { by media handler}

MediaHandler            = ComponentInstance; {media handler}
MediaHandlerComponent    = Component;    {media handler component}
DataHandler             = ComponentInstance; {data handler}
DataHandlerComponent     = Component;    {data handler component}
HandlerError            = ComponentResult; {error handler}

MovieController         = ComponentInstance; {movie controller}

ErrorProcPtr            = ProcPtr;    {pointer to application-defined }
                                { error-notification procedure}

MoviePreviewCallOutProc = ProcPtr;    {pointer to application-defined }
                                { movie preview callout procedure}

TimeBaseFlags           = Char;      {control flags for time base}
QTCallBackProc          = ProcPtr;    {pointer to application-defined }
                                { callback routine}

QTCallBackHeader =
RECORD
    callBackFlags:      LongInt;      {flags used by clock component to }
                                { communicate scheduling data }
                                { about callback to Movie Toolbox}
    reserved1:          LongInt;      {reserved for use by Apple}
    qtPrivate:          PACKED ARRAY[0..39] of Byte;
                                {reserved for use by Apple}
END;

MatrixRecordPtr         = ^MatrixRecord; {pointer to matrix record}

MatrixRecord =
RECORD
    matrix:             ARRAY[0..2,0..2] of Fixed;
END;

FixedPoint =
RECORD

```

## Movie Toolbox

```

    x:          Fixed;    {point's x coordinate as fixed-point number}
    y:          Fixed;    {point's y coordinate as fixed-point number}
END;

FixedRect =
RECORD
    left:      Fixed;    {x coordinate of rectangle's upper-left corner}
    top:       Fixed;    {y coordinate of rectangle's upper-left corner}
    right:     Fixed;    {x coordinate of rectangle's lower-right corner}
    bottom:    Fixed;    {y coordinate of rectangle's lower-right corner}
END;

```

## Routines for Getting and Playing Movies

---

### Initializing the Movie Toolbox

```

FUNCTION EnterMovies:      OSErr;
PROCEDURE ExitMovies;

```

### Error Routines

```

FUNCTION GetMoviesError:   OSErr;
FUNCTION GetMoviesStickyError:
                                OSErr;

PROCEDURE ClearMoviesStickyError;
PROCEDURE SetMoviesErrorProc
                                (errProc: ErrorProcPtr; refcon: LongInt);

```

### Movie Routines

```

FUNCTION NewMovieFromFile   (VAR theMovie: Movie; resRefNum: Integer;
                             VAR resId: Integer; resName: Str255;
                             newMovieFlags: Integer;
                             VAR dataRefWasChanged: Boolean): OSErr;

FUNCTION NewMovieFromHandle
                             (VARh: Handle; newMovieFlags: LongInt;
                             VAR dataRefWasChanged: Boolean): OSErr;

FUNCTION NewMovie           (newMovieFlags: LongInt): Movie;

FUNCTION ConvertFileToMovieFile
                             (inputFile: FSSpec; outputFile: FSSpec;
                             creator: OSType; scriptTag: ScriptCode;
                             VAR resID: Integer; flags: LongInt;
                             userComp: ComponentInstance;
                             proc: ProcPtr; refcon: LongInt): OSErr;

```

## Movie Toolbox

```

FUNCTION ConvertMovieToFile
    (theMovie: Movie; onlyTrack: Track;
     outputFile: FSSpec; fileType: OSType;
     creator: OSType; scriptTag: ScriptCode;
     VAR resID: Integer; flags: LongInt;
     userComp: ComponentInstance): OSErr;

PROCEDURE DisposeMovie
    (theMovie: Movie);

FUNCTION CreateMovieFile
    (fileSpec: FSSpec; creator: OSType;
     scriptTag: ScriptCode;
     createMovieFileFlags: LongInt;
     VAR resRefNum: Integer;
     VAR newMovie: Movie): OSErr;

FUNCTION OpenMovieFile
    (fileSpec: FSSpec; VAR resRefNum: Integer;
     perms: SignedByte): OSErr;

FUNCTION CloseMovieFile
    (resRefNum: Integer): OSErr;

FUNCTION DeleteMovieFile
    (fileSpec: FSSpec): OSErr;

```

**Saving Movies**

```

FUNCTION HasMovieChanged
    (theMovie: Movie): Boolean;

PROCEDURE ClearMovieChanged
    (theMovie: Movie);

FUNCTION AddMovieResource
    (theMovie: Movie; resRefNum: Integer;
     VAR resId: Integer; resName: Str255): OSErr;

FUNCTION UpdateMovieResource
    (theMovie: Movie; resRefNum: Integer;
     VAR resId: Integer; resName: Str255): OSErr;

FUNCTION RemoveMovieResource
    (resRefNum: Integer; resId: Integer): OSErr;

FUNCTION PutMovieIntoHandle
    (theMovie: Movie; publicMovie: Handle): OSErr;

PROCEDURE FlattenMovie
    (theMovie: Movie; movieFlattenFlags: LongInt;
     theFile: FSSpec; creator: OSType;
     scriptTag: ScriptCode;
     createMovieFileFlags: LongInt;
     VAR resId: Integer; resName: Str255);

FUNCTION FlattenMovieData
    (theMovie: Movie; movieFlattenFlags: LongInt;
     theFile: FSSpec; creator: OSType;
     scriptTag: ScriptCode;
     createMovieFileFlags: LongInt): Movie;

```

## Movie Toolbox

```

FUNCTION NewMovieFromDataFork
    (VAR theMovie: Movie; fRefNum: Integer;
     fileOffset: Integer; newMovieFlags: Integer;
     VAR dataRefWasChanged: Boolean): OSErr;

FUNCTION PutMovieIntoDataFork
    (theMovie: Movie; fRefNum: Integer;
     offset: LongInt; maxSize: LongInt): OSErr;

```

**Controlling Movie Playback**

```

PROCEDURE StartMovie      (theMovie: Movie);
PROCEDURE StopMovie       (theMovie: Movie);
PROCEDURE GoToBeginningOfMovie
    (theMovie: Movie);
PROCEDURE GoToEndOfMovie  (theMovie: Movie);

```

**Movie Posters and Movie Previews**

```

PROCEDURE SetTrackUsage    (theTrack: Track; usage: LongInt);
FUNCTION GetTrackUsage     (theTrack: Track): LongInt;
PROCEDURE ShowMoviePoster  (theMovie: Movie);
PROCEDURE SetPosterBox     (theMovie: Movie; boxRect: Rect);
PROCEDURE GetPosterBox     (theMovie: Movie; VAR boxRect: Rect);
PROCEDURE SetMoviePosterTime
    (theMovie: Movie; posterTime: TimeValue);

FUNCTION GetMoviePosterTime
    (theMovie: Movie): TimeValue;

PROCEDURE PlayMoviePreview (theMovie: Movie;
    callOutProc: MoviePreviewCallOutProc;
    refcon: LongInt);

PROCEDURE SetMoviePreviewMode
    (theMovie: Movie; usePreview: Boolean);

FUNCTION GetMoviePreviewMode
    (theMovie: Movie): Boolean;

PROCEDURE SetMoviePreviewTime
    (theMovie: Movie; previewTime: TimeValue;
     previewDuration: TimeValue);

PROCEDURE GetMoviePreviewTime
    (theMovie: Movie; VAR previewTime: TimeValue;
     VAR previewDuration: TimeValue);

```

**Movies and Your Event Loop**

```

PROCEDURE MoviesTask          (theMovie: Movie; maxMilliSecToUse: LongInt);
FUNCTION IsMovieDone          (theMovie: Movie): Boolean;
FUNCTION UpdateMovie          (theMovie: Movie): OSErr;
FUNCTION PtInMovie            (theMovie: Movie; pt: Point): Boolean;
FUNCTION PtInTrack            (theTrack: Track; pt: Point): Boolean;
FUNCTION GetMovieStatus       (theMovie: Movie;
                              VAR firstProblemTrack: Track): ComponentResult;
FUNCTION GetTrackStatus       (theTrack: Track): ComponentResult;

```

**Preferred Movie Settings**

```

PROCEDURE SetMoviePreferredRate
                              (theMovie: Movie; rate: Fixed);

FUNCTION GetMoviePreferredRate
                              (theMovie: Movie): Fixed;

PROCEDURE SetMoviePreferredVolume
                              (theMovie: Movie; volume: Integer);

FUNCTION GetMoviePreferredVolume
                              (theMovie: Movie): Integer;

```

**Enhancing Movie Playback Performance**

```

FUNCTION PrerollMovie          (theMovie: Movie; time: TimeValue;
                              Rate: Fixed): OSErr;

PROCEDURE SetMovieActiveSegment
                              (theMovie: Movie; startTime: TimeValue;
                              duration: TimeValue);

PROCEDURE GetMovieActiveSegment
                              (theMovie: Movie; VAR startTime: TimeValue;
                              VAR duration: TimeValue);

PROCEDURE SetMoviePlayHints    (theMovie: Movie; flags: LongInt;
                              flagsMask: LongInt);

PROCEDURE SetMediaPlayHints    (theMedia: Media; flags: LongInt;
                              flagsMask: LongInt);

FUNCTION LoadMovieIntoRam      (theMovie: Movie; time: TimeValue;
                              duration: TimeValue; flags: LongInt): OSErr;

FUNCTION LoadTrackIntoRam      (theTrack: Track; time: TimeValue;
                              duration: TimeValue; flags: LongInt): OSErr;

FUNCTION LoadMediaIntoRam      (theMedia: Media; time: TimeValue;
                              duration: TimeValue; flags: LongInt): OSErr;

```

## Movie Toolbox

```

FUNCTION SetMediaShadowSync
    (theMedia: Media; frameDiffSampleNum: LongInt;
     syncSampleNum: LongInt): OSErr;

FUNCTION GetMediaShadowSync
    (theMedia: Media; frameDiffSampleNum: LongInt;
     VAR syncSampleNum: LongInt): OSErr;

```

**Disabling Movies and Tracks**

```

PROCEDURE SetMovieActive    (theMovie: Movie; active: Boolean);
FUNCTION GetMovieActive    (theMovie: Movie): Boolean;
PROCEDURE SetTrackEnabled  (theTrack: Track; isEnabled: Boolean);
FUNCTION GetTrackEnabled   (theTrack: Track): Boolean;

```

**Generating Pictures From Movies**

```

FUNCTION GetMoviePict      (theMovie: Movie; time: TimeValue): PicHandle;
FUNCTION GetMoviePosterPict
    (theMovie: Movie): PicHandle;
FUNCTION GetTrackPict      (theTrack: Track; time: TimeValue): PicHandle;

```

**Creating Tracks and Media Structures**

```

FUNCTION NewMovieTrack      (theMovie: Movie; width: Fixed; height: Fixed;
                             trackVolume: Integer): Track;
PROCEDURE DisposeMovieTrack
    (theTrack: Track);
FUNCTION NewTrackMedia      (theTrack: Track; mediaType: OSType;
                             timeScale: TimeScale; dataRef: Handle;
                             dataRefType: OSType): Media;
PROCEDURE DisposeTrackMedia
    (theMedia: Media);

```

**Working With Progress and Cover Procedures**

```

PROCEDURE SetMovieProgressProc
    (theMovie: Movie; p: MovieProgressProcPtr;
     refcon: LongInt);
PROCEDURE SetMovieCoverProcs
    (theMovie: Movie;
     uncoverProc: MovieRgnCoverProc;
     coverProc: MovieRgnCoverProc; refcon: LongInt);

```

## Routines That Modify Movie Properties

---

### Working With Movie Spatial Characteristics

```

PROCEDURE SetMovieGWorld      (theMovie: Movie; port: CGrafPtr;
                               gdh: GDHandle);
PROCEDURE GetMovieGWorld      (theMovie: Movie; VAR port: CGrafPtr;
                               VAR gdh: GDHandle);
PROCEDURE SetMovieBox         (theMovie: Movie; boxRect: Rect);
PROCEDURE GetMovieBox         (theMovie: Movie; VAR boxRect: Rect);
FUNCTION GetMovieDisplayBoundsRgn
                               (theMovie: Movie): RgnHandle;
FUNCTION GetMovieSegmentDisplayBoundsRgn
                               (theMovie: Movie; time: TimeValue;
                               duration: TimeValue): RgnHandle;
PROCEDURE SetMovieDisplayClipRgn
                               (theMovie: Movie; theClip: RgnHandle);
FUNCTION GetMovieDisplayClipRgn
                               (theMovie: Movie): RgnHandle;
FUNCTION GetTrackDisplayBoundsRgn
                               (theTrack: Track): RgnHandle;
FUNCTION GetTrackSegmentDisplayBoundsRgn
                               (theTrack: Track; time: TimeValue;
                               duration: TimeValue): RgnHandle;
PROCEDURE SetTrackLayer       (theTrack: Track; layer: Integer);
FUNCTION GetTrackLayer         (theTrack: Track): Integer;
PROCEDURE SetMovieMatrix      (theMovie: Movie; matrix: MatrixRecord);
PROCEDURE GetMovieMatrix      (theMovie: Movie; VAR matrix: MatrixRecord);
FUNCTION GetMovieBoundsRgn     (theMovie: Movie): RgnHandle;
FUNCTION GetTrackMovieBoundsRgn
                               (theTrack: Track): RgnHandle;
PROCEDURE SetMovieClipRgn     (theMovie: Movie; theClip: RgnHandle);
FUNCTION GetMovieClipRgn      (theMovie: Movie): RgnHandle;
PROCEDURE SetTrackMatrix      (theTrack: Track; matrix: MatrixRecord);
PROCEDURE GetTrackMatrix      (theTrack: Track; VAR matrix: MatrixRecord);
FUNCTION GetTrackBoundsRgn     (theTrack: Track): RgnHandle;
PROCEDURE SetTrackDimensions  (theTrack: Track; width: Fixed; height: Fixed);
PROCEDURE GetTrackDimensions  (theTrack: Track; VAR width: Fixed;
                               VAR height: Fixed);

```



## Movie Toolbox

```

PROCEDURE SetTrackClipRgn    (theTrack: Track; theClip: RgnHandle);
FUNCTION GetTrackClipRgn    (theTrack: Track): RgnHandle;
PROCEDURE SetTrackMatte     (theTrack: Track; theMatte: PixMapHandle);
FUNCTION GetTrackMatte     (theTrack: Track): PixMapHandle;
PROCEDURE DisposeMatte     (theMatte: PixMapHandle);

```

**Working With Sound Volume**

```

PROCEDURE SetMovieVolume    (theMovie: Movie; volume: Integer);
FUNCTION GetMovieVolume    (theMovie: Movie): Integer;
PROCEDURE SetTrackVolume    (theTrack: Track; volume: Integer);
FUNCTION GetTrackVolume    (theTrack: Track): Integer;

```

**Working With Movie Time**

```

FUNCTION GetMovieDuration    (theMovie: Movie): TimeValue;
PROCEDURE SetMovieTimeValue (theMovie: Movie; newtime: TimeValue);

PROCEDURE SetMovieTime      (theMovie: Movie; newtime: TimeRecord);
FUNCTION GetMovieTime      (theMovie: Movie; VAR currentTime: TimeRecord):
    TimeValue;

PROCEDURE SetMovieRate      (theMovie: Movie; rate: Fixed);
FUNCTION GetMovieRate      (theMovie: Movie): Fixed;
PROCEDURE SetMovieTimeScale (theMovie: Movie; timeScale: TimeScale);

FUNCTION GetMovieTimeScale  (theMovie: Movie): TimeScale;
FUNCTION GetMovieTimeBase   (theMovie: Movie): TimeScale;

```

**Working With Track Time**

```

FUNCTION GetTrackDuration    (theTrack: Track): TimeValue;
PROCEDURE SetTrackOffset    (theTrack: Track; movieOffsetTime: TimeValue);
FUNCTION GetTrackOffset    (theTrack: Track): TimeValue;
FUNCTION TrackTimeToMediaTime (value: TimeValue; theTrack: Track): TimeValue;

```

**Working With Media Time**

```

FUNCTION GetMediaDuration    (theMedia: Media): TimeValue;
PROCEDURE SetMediaTimeScale (theMedia: Media; timeScale: TimeScale);

FUNCTION GetMediaTimeScale  (theMedia: Media): TimeScale;

```

## Finding Interesting Times

```
PROCEDURE GetMovieNextInterestingTime
    (theMovie: Movie; interestingTimeFlags: Integer;
     numMediaTypes: Integer;
     whichMediaTypes: OSTypePtr; time: TimeValue;
     rate: Fixed; VAR interestingTime: TimeValue;
     VAR interestingDuration: TimeValue);

PROCEDURE GetTrackNextInterestingTime
    (theTrack: Track; interestingTimeFlags: Integer;
     time: TimeValue; rate: Fixed;
     VAR interestingTime: TimeValue;
     VAR interestingDuration: TimeValue);

PROCEDURE GetMediaNextInterestingTime
    (theMedia: Media; interestingTimeFlags: Integer;
     time: TimeValue; rate: Fixed;
     VAR interestingTime: TimeValue;
     VAR interestingDuration: TimeValue);
```

## Locating a Movie's Tracks and Media Structures

```
FUNCTION GetMovieTrackCount
    (theMovie: Movie): LongInt;

FUNCTION GetMovieIndTrack
    (theMovie: Movie; index: LongInt): Track;

FUNCTION GetMovieTrack
    (theMovie: Movie; trackID: LongInt): Track;

FUNCTION GetTrackID
    (theTrack: Track): LongInt;

FUNCTION GetTrackMovie
    (theTrack: Track): Movie;

FUNCTION GetTrackMedia
    (theTrack: Track): Media;

FUNCTION GetMediaTrack
    (theMedia: Media): Track;
```

## Working With Alternate Tracks

```
PROCEDURE SetMovieLanguage (theMovie: Movie; language: LongInt);

PROCEDURE SelectMovieAlternates
    (theMovie: Movie);

PROCEDURE SetAutoTrackAlternatesEnabled
    (theMovie: Movie; enable: Boolean);

PROCEDURE SetTrackAlternate
    (theTrack: Track; alternateT: Track);

FUNCTION GetTrackAlternate
    (theTrack: Track): Track;

PROCEDURE SetMediaLanguage (theMedia: Media; language: Integer);

FUNCTION GetMediaLanguage
    (theMedia: Media): Integer;

PROCEDURE SetMediaQuality (theMedia: Media; quality: Integer);

FUNCTION GetMediaQuality
    (theMedia: Media): Integer;
```

**Working With Data References**

```

FUNCTION AddMediaDataRef      (theMedia: Media; VAR index: Integer;
                               dataRef: Handle; dataRefType: OSType): OSErr;

FUNCTION SetMediaDataRef      (theMedia: Media; index: Integer;
                               dataRef: Handle; dataRefType: OSType): OSType;

FUNCTION GetMediaDataRefCount  (theMedia: Media; VAR count: Integer): OSErr;

FUNCTION GetMediaDataRef      (theMedia: Media; index: Integer;
                               VAR dataRef: Handle; VAR dataRefType: OSType;
                               VAR dataRefAttributes: LongInt): OSErr;

```

**Determining Movie Creation and Modification Time**

```

FUNCTION GetMovieCreationTime (theMovie: Movie): LongInt;

FUNCTION GetMovieModificationTime (theMovie: Movie): LongInt;

FUNCTION GetTrackCreationTime  (theTrack: Track): LongInt;

FUNCTION GetTrackModificationTime (theTrack: Track): LongInt;

FUNCTION GetMediaCreationTime  (theMedia: Media): LongInt;

FUNCTION GetMediaModificationTime (theMedia: Media): LongInt;

```

**Working With Media Samples**

```

FUNCTION GetMovieDataSize      (theMovie: Movie; startTime: TimeValue;
                               duration: TimeValue): LongInt;

FUNCTION GetTrackDataSize      (theTrack: Track; startTime: TimeValue;
                               duration: TimeValue): LongInt;

FUNCTION GetMediaDataSize      (theMedia: Media; startTime: TimeValue;
                               duration: TimeValue): LongInt;

FUNCTION GetMediaSampleCount    (theMedia: Media): LongInt;

FUNCTION GetMediaSampleDescriptionCount (theMedia: Media): LongInt;

PROCEDURE GetMediaSampleDescription (theMedia: Media; index: LongInt;
                                      descH: SampleDescriptionHandle);

FUNCTION SetMediaSampleDescription (theMedia: Media; index: LongInt;
                                      descH: SampleDescriptionHandle): OSErr;

```

## Movie Toolbox

```

PROCEDURE MediaTimeToSampleNum
    (theMedia: Media; time: TimeValue;
     VAR sampleNum: LongInt;
     VAR sampleTime: TimeValue;
     VAR sampleDuration: TimeValue);

PROCEDURE SampleNumToMediaTime
    (theMedia: Media; logicalSampleNum: LongInt;
     VAR sampleTime: TimeValue;
     VAR sampleDuration: TimeValue);

```

**Working With Movie User Data**

```

FUNCTION GetMovieUserData    (theMovie: Movie): UserData;
FUNCTION GetTrackUserData    (theTrack: Track): UserData;
FUNCTION GetMediaUserData    (theMedia: Media): UserData;
FUNCTION GetNextUserDataType
    (theUserData: UserData;
     udType: OSType): LongInt;

FUNCTION CountUserDataTypes  (theUserData: UserData;
                               udType: OSType): Integer;

FUNCTION AddUserData          (theUserData: UserData; data: Handle;
                               udType: OSType): OSErr;

FUNCTION GetUserData          (theUserData: UserData; data: Handle;
                               udType: OSType; index: LongInt): OSErr;

FUNCTION RemoveUserData       (theUserData: UserData; udType: OSType;
                               index: LongInt): OSErr;

FUNCTION AddUserDataText      (theUserData: UserData; data: Handle;
                               udType: OSType;
                               index: LongInt; itlRegionTag: Integer): OSErr;

FUNCTION GetUserDataText      (theUserData: UserData; data: Handle;
                               udType: OSType; index: LongInt;
                               itlRegionTag: Integer): OSErr;

FUNCTION RemoveUserDataText    (theUserData: UserData; udType: OSType;
                               index: LongInt; itlRegionTag: Integer): OSErr;

FUNCTION SetUserDataItem      (theUserData: UserData; data: Ptr;
                               size: LongInt; udType: OSType;
                               index: LongInt): OSErr;

FUNCTION GetUserDataItem      (theUserData: UserData; data: Ptr;
                               size: LongInt; udType: OSType; index: long):
    OSErr;

FUNCTION NewUserData          (VAR theUserData: UserData): OSErr;
FUNCTION DisposeUserData      (theUserData: UserData): OSErr;

```

## Movie Toolbox

```

FUNCTION PutUserDataIntoHandle
    (theUserData UserData; h Handle): OSErr;

FUNCTION NewUserDataFromHandle
    (h Handle; VAR theUserData: UserData): OSErr;

```

## Routines for Editing Movies

---

### Editing Movies

```

FUNCTION PutMovieOnScrap    (theMovie: Movie;
    movieScrapFlags: LongInt): OSErr;

FUNCTION NewMovieFromScrap (newMovieFlags: LongInt): Movie;

PROCEDURE SetMovieSelection
    (theMovie: Movie; selectionTime: TimeValue;
    selectionDuration: TimeValue);

PROCEDURE GetMovieSelection
    (theMovie: Movie; VAR selectionTime: TimeValue;
    VAR selectionDuration: TimeValue);

FUNCTION CutMovieSelection (theMovie: Movie): Movie;

FUNCTION CopyMovieSelection
    (theMovie: Movie): Movie;

PROCEDURE PasteMovieSelection
    (theMovie: Movie; src: Movie);

PROCEDURE AddMovieSelection
    (theMovie: Movie; src: Movie);

PROCEDURE ClearMovieSelection
    (theMovie: Movie);

FUNCTION IsScrapMovie      (targetTrack: Track): Component;

FUNCTION PasteHandleIntoMovie
    (h: Handle; handleType: OSType;
    theMovie: Movie; flags: LongInt;
    userComp: ComponentInstance): OSErr;

FUNCTION PutMovieIntoTypedHandle
    (theMovie: Movie; targetTrack: Track;
    handleType: OSType; publicMovie: Handle;
    start: TimeValue; dur: TimeValue;
    flags: long; userComp: ComponentInstance):
    OSErr;

```

**Undo for Movies**

```

FUNCTION NewMovieEditState (theMovie: Movie): MovieEditState;
FUNCTION UseMovieEditState (theMovie: Movie; toState: MovieEditState):
    OSErr;
FUNCTION DisposeMovieEditState
    (state: MovieEditState): OSErr;

```

**Low-Level Movie-Editing Routines**

```

FUNCTION InsertMovieSegment (srcMovie: Movie; dstMovie: Movie;
    srcIn: TimeValue; srcDuration: TimeValue;
    dstIn: TimeValue): OSErr;
FUNCTION InsertEmptyMovieSegment
    (dstMovie: Movie; dstIn: TimeValue;
    dstDuration: TimeValue): OSErr;
FUNCTION DeleteMovieSegment
    (theMovie: Movie; in: TimeValue;
    duration: TimeValue): OSErr;
FUNCTION ScaleMovieSegment (theMovie: Movie; in: TimeValue;
    oldDuration: TimeValue;
    newDuration: TimeValue): OSErr;
FUNCTION CopyMovieSettings (srcMovie: Movie; dstMovie: Movie): OSErr;

```

**Editing Tracks**

```

FUNCTION InsertTrackSegment
    (srcTrack: Track; dstTrack: Track;
    srcIn: TimeValue; srcDuration: TimeValue;
    dstIn: TimeValue): OSErr;
FUNCTION InsertEmptyTrackSegment
    (dstTrack: Track; dstIn: TimeValue;
    dstDuration: TimeValue): OSErr;
FUNCTION InsertMediaIntoTrack
    (theTrack: Track; trackStart: TimeValue;
    mediaTime: TimeValue; mediaDuration: TimeValue;
    mediaRate: Fixed): OSErr;
FUNCTION DeleteTrackSegment
    (theTrack: Track; in: TimeValue;
    duration: TimeValue): OSErr;

```

## Movie Toolbox

```

FUNCTION ScaleTrackSegment (theTrack: Track; in: TimeValue;
                           oldDuration: TimeValue;
                           newDuration: TimeValue): OSErr;

FUNCTION CopyTrackSettings (srcTrack: Track; dstTrack: Track): OSErr;

FUNCTION GetTrackEditRate (theTrack: Track; atTime: TimeValue): Fixed;

```

**Undo for Tracks**

```

FUNCTION NewTrackEditState (theTrack: Track): TrackEditState;

FUNCTION UseTrackEditState (theTrack: Track; state: TrackEditState): OSErr;

FUNCTION DisposeTrackEditState
                           (state: TrackEditState): OSErr;

```

**Adding Samples to Media Structures**

```

FUNCTION BeginMediaEdits (theMedia: Media): OSErr;

FUNCTION EndMediaEdits (theMedia: Media): OSErr;

FUNCTION AddMediaSample (theMedia: Media; dataIn: Handle;
                        inOffset: LongInt; size: LongInt;
                        durationPerSample: TimeValue;
                        sampleDescriptionH: SampleDescriptionHandle;
                        numberOfSamples: LongInt;
                        sampleFlags: Integer;
                        VAR sampleTime: TimeValue): OSErr;

FUNCTION AddMediaSampleReference
                           (theMedia: Media; dataOffset: LongInt;
                           size: LongInt; durationPerSample: TimeValue;
                           sampleDescriptionH: SampleDescriptionHandle;
                           numberOfSamples: LongInt;
                           sampleFlags: Integer;
                           VAR sampleTime: TimeValue): OSErr;

FUNCTION GetMediaSample (theMedia: Media; dataOut: Handle;
                        maxSizeToGrow: LongInt; VAR size: LongInt;
                        time: TimeValue;
                        VAR sampleTime: TimeValue;
                        VAR durationPerSample: TimeValue;
                        sampleDescriptionH: SampleDescriptionHandle;
                        VAR sampleDescriptionIndex: LongInt;
                        maxNumberOfSamples: LongInt;
                        VAR numberOfSamples: LongInt;
                        VAR sampleFlags: Integer): OSErr;

```

```
FUNCTION GetMediaSampleReference
```

```
    (theMedia: Media; VAR dataOffset: LongInt;
     VAR size: LongInt; time: TimeValue;
     VAR sampleTime: TimeValue;
     VAR durationPerSample: TimeValue;
     sampleDescriptionH: SampleDescriptionHandle;
     VAR sampleDescriptionIndex: LongInt;
     maxNumberOfSamples: LongInt;
     VAR numberOfSamples: LongInt;
     VAR sampleFlags: Integer): OSErr;
```

## Media Routines

---

### Selecting Media Handlers

```
PROCEDURE GetMediaHandlerDescription
```

```
    (theMedia: Media; VAR mediaType: OSType;
     VAR creatorName: Str255;
     VAR creatorManufacturer: OSType);
```

```
FUNCTION GetMediaHandler    (theMedia: Media): MediaHandler;
```

```
FUNCTION SetMediaHandler    (theMedia: Media;
                             mh: MediaHandlerComponent): OSErr;
```

```
PROCEDURE GetMediaDataHandlerDescription
```

```
    (theMedia: Media; index: Integer;
     VAR dhType: OSType;
     VAR creatorName: Str255;
     VAR creatorManufacturer: OSType);
```

```
FUNCTION GetMediaDataHandler
```

```
    (theMedia: Media; index: Integer): DataHandler;
```

```
FUNCTION SetMediaDataHandler
```

```
    (theMedia: Media; index: Integer;
     dataHandler: DataHandlerComponent): OSErr;
```

### Video Media Handler Routines

```
FUNCTION SetVideoMediaGraphicsMode
```

```
    (mh: MediaHandler; graphicsMode: LongInt;
     opColor: RGBColor): HandlerError;
```

```
FUNCTION GetVideoMediaGraphicsMode
```

```
    (mh: MediaHandler; VAR graphicsMode: LongInt;
     VAR opColor: RGBColor): HandlerError;
```



**Sound Media Handler Routines**

```

FUNCTION SetSoundMediaBalance
    (mh: MediaHandler;
     balance: Integer): HandlerError;

FUNCTION GetSoundMediaBalance
    (mh: MediaHandler;
     VAR balance: Integer): HandlerError;

```

**Text Media Handler Routines**

```

FUNCTION AddTextSample
    (mh: MediaHandler; text: Ptr; size: LongInt;
     fontNumber: Integer; fontSize: Integer;
     textFace: Style; textColor: RGBColor;
     backColor: RGBColor;
     textJustification: Integer; VAR textBox: Rect;
     displayFlags: LongInt;
     scrollDelay: TimeValue;
     hiliteStart: Integer; hiliteEnd: Integer;
     VAR rgbColor: RGBColor;
     duration: TimeValue;
     VAR sampleTime: TimeValue): ComponentResult;

FUNCTION AddTESample
    (mh: MediaHandler; hTE: TEHandle;
     VAR backColor: RGBColor;
     textJustification: Integer; VAR textBox: Rect;
     displayFlags: LongInt;
     scrollDelay: TimeValue;
     hiliteStart: Integer; hiliteEnd: Integer;
     VAR rgbColor: RGBColor;
     duration: TimeValue;
     VAR sampleTime: TimeValue): ComponentResult;

FUNCTION AddHiliteSample
    (mh: MediaHandler; hiliteStart: Integer;
     hiliteEnd: Integer; VAR rgbColor: RGBColor;
     duration: TimeValue;
     VAR sampleTime: TimeValue): ComponentResult;

FUNCTION FindNextText
    (mh: MediaHandler; text: Ptr; size: LongInt;
     findFlags: Integer; startTime: TimeValue;
     VAR foundTime: TimeValue;
     VAR foundDuration: TimeValue;
     VAR offset: LongInt): ComponentResult;

FUNCTION HiliteTextSample
    (mh: MediaHandler; sampleTime: TimeValue;
     hiliteStart: Integer;
     hiliteEnd: Integer;
     VAR rgbHiliteColor: RGBColor): ComponentResult;

FUNCTION SetTextProc
    (mh: MediaHandler; TextProc: ProcPtr;
     refcon: LongInt): ComponentResult;

```

## Routines for Creating File Previews

---

```

FUNCTION MakeFilePreview    (resRefNum: Integer;
                             progress: ProgressProcRecordPtr): OSErr;

FUNCTION AddFilePreview     (resRefNum: Integer; previewType: OSType;
                             previewData: Handle): OSErr;

```

## Routines for Displaying File Previews

---

```

PROCEDURE SFGetFilePreview  (where: Point; prompt: Str255;
                             fileFilter: FileFilterProcPtr;
                             numTypes: Integer; typeList: SFTypeList;
                             dlgHook: DlgHookProcPtr; VAR reply: SFReply);

PROCEDURE SFPGetFilePreview
                             (where: Point; prompt: Str255;
                             fileFilter: FileFilterProcPtr;
                             numTypes: Integer; typeList: SFTypeList;
                             dlgHook: DlgHookProcPtr; VAR reply: SFReply;
                             dlgID: Integer; filterProc:
                             ModalFilterProcPtr);

PROCEDURE StandardGetFilePreview
                             (fileFilter: FileFilterProcPtr;
                             numTypes: Integer; typeList: SFTypeList;
                             VAR reply: StandardFileReply);

PROCEDURE CustomGetFilePreview
                             (fileFilter: FileFilterYDProcPtr;
                             numTypes: Integer; typeList: SFTypeList;
                             VAR reply: StandardFileReply; dlgID: Integer;
                             where: Point; dlgHook: DlgHookYDProcPtr;
                             filterProc: ModalFilterYDProcPtr;
                             activeList: Ptr;
                             activateProc: ActivateYDProcPtr;
                             yourDataPtr: UNIV Ptr);

```

## Time Base Routines

---

### Creating and Disposing of Time Bases

```

FUNCTION NewTimeBase:       TimeBase;

PROCEDURE DisposeTimeBase  (tb: TimeBase);

PROCEDURE SetMovieMasterClock
                             (theMovie: Movie; clockMeister: Component;
                             slaveZero: TimeRecord);

```

## Movie Toolbox

```

PROCEDURE SetMovieMasterTimeBase
    (theMovie: Movie; tb: TimeBase;
     slaveZero: TimeRecord);

PROCEDURE SetTimeBaseMasterClock
    (slave: TimeBase; clockMeister: Component;
     slaveZero: TimeRecord);

FUNCTION GetTimeBaseMasterClock
    (tb: TimeBase): ComponentInstance;

PROCEDURE SetTimeBaseMasterTimeBase
    (slave: TimeBase; master: TimeBase;
     slaveZero: TimeRecord);

FUNCTION GetTimeBaseMasterTimeBase
    (tb: TimeBase): TimeBase;

PROCEDURE SetTimeBaseZero    (tb: TimeBase; VAR zero: TimeRecord);

```

**Working With Time Base Values**

```

PROCEDURE SetTimeBaseTime    (tb: TimeBase; tr: TimeRecord);
PROCEDURE SetTimeBaseValue   (tb: TimeBase; t: TimeValue; s: TimeScale);
FUNCTION GetTimeBaseTime     (tb: TimeBase; s: TimeScale;
                             VAR tr: TimeRecord): TimeValue;

PROCEDURE SetTimeBaseRate    (tb: TimeBase; r: Fixed);
FUNCTION GetTimeBaseRate     (tb: TimeBase): Fixed;
FUNCTION GetTimeBaseEffectiveRate
    (tb: TimeBase): Fixed;

PROCEDURE SetTimeBaseStartTime
    (tb: TimeBase; VAR tr: TimeRecord);

FUNCTION GetTimeBaseStartTime
    (tb: TimeBase; s: TimeScale;
     tr: TimeRecord): TimeValue;

PROCEDURE SetTimeBaseStopTime
    (tb: TimeBase; VAR tr: TimeRecord);

FUNCTION GetTimeBaseStopTime
    (tb: TimeBase; s: TimeScale;
     tr: TimeRecord): TimeValue;

PROCEDURE SetTimeBaseFlags   (tb: TimeBase; timeBaseFlags: LongInt);
FUNCTION GetTimeBaseFlags    (tb: TimeBase): LongInt;
FUNCTION GetTimeBaseStatus    (tb: TimeBase;
                             VAR unpinnedTime: TimeRecord): LongInt;

```

**Working With Times**

```

PROCEDURE AddTime          (VAR dst: TimeRecord; src: TimeRecord);
PROCEDURE SubtractTime     (VAR dst: TimeRecord; src: TimeRecord);
PROCEDURE ConvertTime      (VAR inout: TimeRecord; newBase: TimeBase);
PROCEDURE ConvertTimeScale (VAR inout: TimeRecord; newScale: TimeScale);

```

**Time Base Callback Routines**

```

FUNCTION NewCallBack      (tb: TimeBase; cbType: Integer): QTCallBack;
FUNCTION CallMeWhen       (cb: QTCallBack; callBackProc: QTCallBackProc;
                           refcon: LongInt; param1: LongInt;
                           param2: LongInt; param3: LongInt): OSErr;

PROCEDURE CancelCallBack  (cb: QTCallBack);
PROCEDURE DisposeCallBack (cb: QTCallBack);
FUNCTION GetCallBackTimeBase
                           (cb: QTCallBack): TimeBase;
FUNCTION GetCallBackType   (cb: QTCallBack): Integer;

```

**Matrix Routines**


---

```

PROCEDURE SetIdentityMatrix
                           (VAR matrix: MatrixRecord);

FUNCTION GetMatrixType     (m: MatrixRecord): Integer;
PROCEDURE CopyMatrix       (m1: MatrixRecord; VAR m2: MatrixRecord);
FUNCTION EqualMatrix       (m1: MatrixRecord; m2: MatrixRecord): Boolean;
PROCEDURE TranslateMatrix  (VAR m: MatrixRecord; deltaH: Fixed;
                           deltaV: Fixed);

PROCEDURE ScaleMatrix      (VAR m: MatrixRecord; scaleX: Fixed;
                           scaleY: Fixed; aboutX: Fixed; aboutY: Fixed);
PROCEDURE RotateMatrix     (VAR m: MatrixRecord; degrees: Fixed;
                           aboutX: Fixed; aboutY: Fixed);
PROCEDURE SkewMatrix       (VAR m: MatrixRecord; skewX: Fixed;
                           skewY: Fixed; aboutX: Fixed; aboutY: Fixed);
PROCEDURE ConcatMatrix     (a: MatrixRecord; VAR b: MatrixRecord);
FUNCTION InverseMatrix      (m: MatrixRecord;
                           VAR im: MatrixRecord): Boolean;
FUNCTION TransformPoints    (mp: MatrixRecord; VAR pt1: Point;
                           count: LongInt): OSErr;
FUNCTION TransformFixedPoints
                           (m: MatrixRecord; VAR fpt: FixedPoint;
                           count: LongInt): OSErr;

```

## Movie Toolbox

```

FUNCTION TransformRect      (m: MatrixRecord; VAR r: Rect;
                           VAR fpp: FixedPoint): Boolean;

FUNCTION TransformFixedRect (m: MatrixRecord; VAR fr: FixedRect;
                           VAR fpp: FixedPoint): Boolean;

FUNCTION TransformRgn      (mp: MatrixRecord; r: RgnHandle): OSErr;

PROCEDURE RectMatrix      (VAR matrix: MatrixRecord; srcRect: Rect;
                           dstRect: Rect);

PROCEDURE MapMatrix      (VAR matrix: MatrixRecord; fromRect: Rect;
                           toRect: Rect);

```

Application-Defined Routines

---

**Progress Routines**

```

FUNCTION MyProgressProc      (theMovie: Movie; message: Integer;
                           whatOperation: Integer;
                           percentDone: Fixed; refcon: LongInt): OSErr;

```

**Cover Routines**

```

FUNCTION MyCoverProc      (theMovie: Movie; changedRgn: RgnHandle;
                           refcon: LongInt): OSErr;

```

**Error-Notification Routines**

```

PROCEDURE MyErrProc      (theErr: OSErr; refcon: LongInt);

```

**Movie Callout Routines**

```

FUNCTION MyCallOutProc      (refcon: long): Boolean;

```

**File Filter Routines**

```

FUNCTION MyFileFilter      (paramBlock: ParmBlkPtr): Boolean;

```

**Custom Routines**

```

FUNCTION MyDlgHook      (item: Integer; theDialog: DialogPtr;
                           myDataPtr: Ptr): Integer;

```

**Modal-Dialog Filter Routines**

```

FUNCTION MyModalFilter      (theDialog: DialogPtr;
                           VAR theEvent: EventRecord; itemHit: Integer;
                           myDataPtr: Ptr): Boolean;

```

**Standard File Activation Routines**

```
PROCEDURE MyActivateProc      (theDialog: DialogPtr; itemNo: Integer;
                               activating: Boolean; myDataPtr: Ptr);
```

**Callback Event Routines**

```
PROCEDURE MyCallBackProc      (cb: QTCallBack; refcon: LongInt);
```

**Text Routines**

```
PROCEDURE MyTextProc          (theText: Handle; theMovie: Movie;
                               VAR displayFlag: Integer; refcon: LongInt);
```

**Result Codes**


---

couldNotResolveDataRef	-2000	Cannot use this data reference
badImageDescription	-2001	Problem with this image description
badPublicMovieAtom	-2002	Movie file corrupted
cantFindHandler	-2003	Cannot locate this handler
cantOpenHandler	-2004	Cannot open this handler
badComponentType	-2005	Component cannot accommodate this data
noMediaHandler	-2006	Media has no media handler
noDataHandler	-2007	Media has no data handler
invalidMedia	-2008	This media is corrupted or invalid
invalidTrack	-2009	This track is corrupted or invalid
invalidMovie	-2010	This movie is corrupted or invalid
invalidSampleTable	-2011	This sample table is corrupted or invalid
invalidDataRef	-2012	This data reference is invalid
invalidHandler	-2013	This handler is invalid
invalidDuration	-2014	This duration value is invalid
invalidTime	-2015	This time value is invalid
cantPutPublicMovieAtom	-2016	Cannot write to this movie file
badEditList	-2017	The track's edit list is corrupted
mediaTypesDontMatch	-2018	These media don't match
progressProcAborted	-2019	Your progress procedure returned an error
movieToolboxUninitialized	-2020	You haven't initialized the Movie Toolbox
wfFileNotFound	-2021	Cannot locate this file
cantCreateSingleForkFile	-2022	Error trying to create a single-fork file. This occurs when the file already exists.
invalidEditState	-2023	This edit state is invalid
nonMatchingEditState	-2024	This edit state is not valid for this movie
staleEditState	-2025	Movie or track has been disposed
userDataItemNotFound	-2026	Cannot locate this user data item
maxSizeToGrowTooSmall	-2027	Maximum size must be larger
badTrackIndex	-2028	This track index value is not valid
trackIDNotFound	-2029	Cannot locate a track with this ID value
trackNotInMovie	-2030	This track is not in this movie
timeNotInTrack	-2031	This time value is outside of this track
timeNotInMedia	-2032	This time value is outside of this media

## Movie Toolbox

badEditIndex	-2033	This edit index value is not valid
internalQuickTimeError	-2034	Internal error
cantEnableTrack	-2035	Cannot enable this track
invalidRect	-2036	Specified rectangle has invalid coordinates
invalidSampleNum	-2037	There is no sample with this sample number
invalidChunkNum	-2038	There is no chunk with this chunk number
invalidSampleDescIndex	-2039	Sample description index value invalid
invalidChunkCache	-2040	The chunk cache is corrupted
invalidSampleDescription	-2041	This sample description is invalid or corrupted
dataNotOpenForRead	-2042	Cannot read from this data source
dataNotOpenForWrite	-2043	Cannot write to this data source
dataAlreadyOpenForWrite	-2044	Data source is already open for write
dataAlreadyClosed	-2045	You have already closed this data source
endOfDataReached	-2046	End of data
dataNoDataRef	-2047	No data reference value found
noMovieFound	-2048	Toolbox cannot find a movie in the movie file
invalidDataRefContainer	-2049	Invalid data reference
badDataRefIndex	-2050	Data reference index value is invalid
noDefaultDataRef	-2051	Could not find a default data reference
couldNotUseAnExistingSample	-2052	Movie Toolbox could not use a sample
featureUnsupported	-2053	Movie Toolbox does not support this feature