QuickTime Callbacks Reference

QuickTime

2006-05-23

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

Simultaneously published in the United States and Canada.

Contents

QuickTime Callbacks Reference

Framework: Frameworks/QuickTime.framework
Declared in QuickTime.h

Overview

This reference covers the callbacks common to multiple QuickTime frameworks.

Callbacks

ActionsProc

Action callback for a media handler.

```
typedef OSErr (*ActionsProcPtr) (void *refcon, Track targetTrack, long targetRefCon,
 QTEventRecordPtr theEvent);
```

If you name your function `MyActionsProc`, you would declare it this way:

```
OSErr MyActionsProc (
    void                *refcon,
    Track               targetTrack,
    long                targetRefCon,
    QTEventRecordPtr    theEvent );
```

Parameters

refcon
: A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

targetTrack
: The track in which to perform the actions.

targetRefCon
: A reference constant for the target track.

theEvent
: Pointer to a `QTEventRecord` structure.

Return Value

See `Error Codes`. Your callback should return `noErr` if there is no error.

See Also

See the `MediaSetActionsCallback` and `NewActionsUPP` functions.

DoMCActionProc

Undocumented

```
typedef OSErr (*DoMCActionProcPtr) (void *refcon, short action, void *params,
Boolean *handled);
```

If you name your function `MyDoMCActionProc`, you would declare it this way:

```
OSErr MyDoMCActionProc (
    void       *refcon,
    short      action,
    void       *params,
    Boolean    *handled );
```

Parameters

refcon

Pointer to a reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

action

Undocumented

params

Undocumented

handled

A pointer to a Boolean in which you put TRUE if the action was handled, FALSE otherwise.

Return Value

See `Error Codes`. Your callback should return `noErr` if there is no error.

See Also

See the `MCGetDoActionsProc`, `MediaSetDoMCActionCallback`, `MovieMediaGetDoMCActionCallback`, `MovieMediaGetChildDoMCActionCallback`, and `NewDoMCActionUPP` functions.

ICMConvertDataFormatProc

Undocumented

```
typedef OSErr (*ICMConvertDataFormatProcPtr) (void *refCon, long flags, Handle
desiredFormat, Handle sourceDataFormat, void *srcData, long srcDataSize,
void **dstData, long *dstDataSize);
```

If you name your function `MyICMConvertDataFormatProc`, you would declare it this way:

```
OSErr MyICMConvertDataFormatProc (
    void      *refCon,
    long      flags,
    Handle    desiredFormat,
    Handle    sourceDataFormat,
    void      *srcData,
```

```
    long      srcDataSize,
    void      **dstData,
    long      *dstDataSize );
```

Parameters

refCon

Pointer to a reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

flags

Undocumented

desiredFormat

Undocumented

sourceDataFormat

Undocumented

srcData

Undocumented

srcDataSize

Undocumented

dstData

Undocumented

dstDataSize

Undocumented

Return Value

See `Error Codes`. Your callback should return `noErr` if there is no error.

See Also

See the `CDSequenceNewDataSource` and `NewICMConvertDataFormatUPP` functions.

ICMMemoryDisposedProc

Called before disposing of the memory allocated by a codec.

```
typedef void (*ICMMemoryDisposedProcPtr) (Ptr memoryBlock, void *refcon);
```

If you name your function `MyICMMemoryDisposedProc`, you would declare it this way:

```
void MyICMMemoryDisposedProc (
    Ptr     memoryBlock,
    void    *refcon );
```

Parameters

memoryBlock

Pointer to a block of memory.

refcon

Contains a reference constant value that your codec must pass to the `memoryGoneProc` function.

Discussion
This function must be called if the memory block is to be disposed of by the codec instead of by `ImageCodecDisposeMemory`. For example, this would occur if the codec is closed and still has memory allocation outstanding or if the memory is required to complete another operation.

Special Considerations

The Image Compression Manager does not currently track memory allocations. When a compressor or decompressor component instance is closed, it must ensure that all blocks allocated by that instance are disposed and call your `ICMMemoryDisposedProc`. This callback must not be called at interrupt time.

See Also

See the `CDSequenceNewMemory`, `ImageCodecNewImageBufferMemory`, `ImageCodecNewMemory`, and `NewICMMemoryDisposedUPP` functions.

ModalFilterProc

Determines how events are filtered for modal dialog boxes.

```
typedef Boolean (*ModalFilterProcPtr) (DialogPtr theDialog, EventRecord *theEvent,
 DialogItemIndex *itemHit);
```

If you name your function `MyModalFilterProc`, you would declare it this way:

```
Boolean MyModalFilterProc (
    DialogPtr          theDialog,
    EventRecord        *theEvent,
    DialogItemIndex    *itemHit );
```

Parameters

`theDialog`

A pointer to the dialog record.

`theEvent`

A pointer to the event record.

`itemHit`

The item number.

Return Value
Your `ModalFilterProc` callback returns a Boolean value that reports whether it handled the event. If your function returns a value of FALSE, QuickTime processes the event through its own filters. If your function returns a value of TRUE, QuickTime returns with no further action.

Discussion
The Standard File Package contains an internal filter function that performs some preliminary processing on each event it receives. If you provide a `ModalFilterProc` callback, it is called after the internal Standard File Package filter function and before the event is sent to your dialog hook function. You might provide a `ModalFilterProc` callback for several reasons. If you have customized the Open or Save dialog boxes by adding one or more items, you might want to map some of the user's keypresses to those items in the same way that the internal filter function maps certain keypresses to existing items.

Special Considerations

You can supply a `ModalFilterProc` callback only when you use one of the procedures that displays a customized dialog box. Another reason to provide a `ModalFilterProc` callback is to avoid a problem that can arise if an update event is received for one of your application's windows while a Standard File Package dialog box is displayed.

See Also

See the `ImageCodecRequestSettings`, `QTVideoOutputCustomConfigureDisplay`, `SFPGetFilePreview`, and similar functions.

MovieDrawingCompleteProc

Called when movie drawing is complete.

```
typedef OSErr (*MovieDrawingCompleteProcPtr) (Movie theMovie, long refCon);
```

If you name your function `MyMovieDrawingCompleteProc`, you would declare it this way:

```
OSErr MyMovieDrawingCompleteProc (
    Movie    theMovie,
    long     refCon );
```

Parameters

`theMovie`

Specifies the movie for this operation.

`refCon`

Contains the reference constant you supplied when your application called the `SetMovieDrawingCompleteProc` function.

Return Value

See `Error Codes`. Your callback should return `noErr` if there is no error.

Special Considerations

Some media handlers may take less efficient playback paths when a `MovieDrawingCompleteProc` is used, so it should be used only when absolutely necessary.

See Also

See the `SetMovieDrawingCompleteProc` and `NewMovieDrawingCompleteUPP` functions.

MoviePrePrerollCompleteProc

Undocumented

```
typedef void (*MoviePrePrerollCompleteProcPtr) (Movie theMovie, OSErr prerollErr,
 void *refcon);
```

If you name your function `MyMoviePrePrerollCompleteProc`, you would declare it this way:

```
void MyMoviePrePrerollCompleteProc (
    Movie    theMovie,
    OSErr    prerollErr,
```

```
    void     *refcon );
```

Parameters

theMovie

Specifies the movie for this operation.

prerollErr

Undocumented

refcon

Pointer to a reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

See Also

See the `PrePrerollMovie` and `NewMoviePrePrerollCompleteUPP` functions.

MoviePreviewCallOutProc

Controls the playing of a movie's preview.

```
typedef Boolean (*MoviePreviewCallOutProcPtr) (long refcon);
```

If you name your function `MyMoviePreviewCallOutProc`, you would declare it this way:

```
Boolean MyMoviePreviewCallOutProc (
    long    refcon );
```

Parameters

refcon

A reference constant you specified when you called `PlayMoviePreview`.

Return Value

If your function sets this value to FALSE, the Movie Toolbox stops the preview and returns to your application.

Discussion

If you call `GetMovieActiveSegment` from within your callback, the Movie Toolbox can change the active movie segment to be the preview segment of the movie. The Movie Toolbox will restore the active segment when the preview is done playing.

See Also

See `PlayMoviePreview`.

MovieProgressProc

Monitors the progress of the Movie Toolbox during long operations.

```
typedef OSErr (*MovieProgressProcPtr) (Movie theMovie, short message, short
whatOperation, Fixed percentDone, long refcon);
```

If you name your function `MyMovieProgressProc`, you would declare it this way:

```
OSErr MyMovieProgressProc (
    Movie    theMovie,
```

```
    short    message,
    short    whatOperation,
    Fixed    percentDone,
    long     refcon );
```

Parameters

theMovie

Specifies the movie for this operation. The Movie Toolbox sets this parameter to identify the appropriate movie.

message

Constant (see below) that indicates why the Movie Toolbox called your function. See these constants:

```
movieProgressOpen
movieProgressUpdatePercent
movieProgressClose
```

whatOperation

Constant (see below) that indicates the long operation that is currently underway. See these constants:

```
progressOpFlatten
progressOpInsertTrackSegment
progressOpInsertMovieSegment
progressOpPaste
progressOpAddMovieSelection
progressOpCopy
progressOpCut
progressOpLoadMovieIntoRam
progressOpLoadTrackIntoRam
progressOpLoadMediaIntoRam
progressOpImportMovie
progressOpExportMovie
```

percentDone

Contains a fixed-point value indicating how far the operation has progressed. Its value is always between 0.0 and 1.0. This parameter is valid only when the `message` field is set to `movieProgressUpdatePercent`.

refcon

Reference constant value for use by your progress function. Your application specifies the value of this reference constant when you assign the progress function to the movie.

Return Value

Your progress function should return an error value. The Movie Toolbox examines this value after each `movieProgressUpdatePercent` message and before continuing the current operation. Set this value to a nonzero value to cancel the operation; set it to `noErr` to continue.

See Also

See the `ConvertFileToMovieFile`, `GetMovieProgressProc`, `MovieExportSetProgressProc`, `MovieImportSetProgressProc`, `SetMovieProgressProc`, and `NewMovieProgressUPP` functions.

MoviesErrorProc

An error-notification function, called each time the current error value is to be set to a nonzero value.

```
typedef void (*MoviesErrorProcPtr) (OSErr theErr, long refcon);
```

If you name your function `MyMoviesErrorProc`, you would declare it this way:

```
void MyMoviesErrorProc (
    OSErr    theErr,
    long     refcon );
```

Parameters

theErr

An error code; see `Error Codes`.

refcon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

See Also

See the `SetMoviesErrorProc` and `NewMoviesErrorUPP` functions.

QTCallBackProc

A generic callback function, installed by CallMeWhen.

```
typedef void (*QTCallBackProcPtr) (QTCallBack cb, long refCon);
```

If you name your function `MyQTCallBackProc`, you would declare it this way:

```
void MyQTCallBackProc (
    QTCallBack    cb,
    long          refCon );
```

Parameters

cb

A pointer to a `CallBackRecord` structure containing the callback's data.

refCon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

See Also

See the `CallMeWhen` and `NewQTCallBackUPP` functions.

QTNextTaskNeededSoonerCallbackProc

Called when QuickTime decides that the next task is needed sooner than previously reported by QTGetTimeUntilNextTask.

```
typedef void (*QTNextTaskNeededSoonerCallbackProcPtr) (TimeValue duration, unsigned
 long flags, void *refcon);
```

If you name your function `MyQTNextTaskNeededSoonerCallbackProc`, you would declare it this way:

```
void MyQTNextTaskNeededSoonerCallbackProc (
    TimeValue        duration,
    unsigned long    flags,
    void             *refcon );
```

Parameters

duration

The new duration to the next task, expressed in the time scale set by a previous call to `QTInstallNextTaskNeededSoonerCallback`.

flags

Undocumented

refcon

A pointer to a reference constant. You can use this parameter to point to a data structure containing any information your callback may need.

Return Value

See `Error Codes`. Your callback should return `noErr` if there is no error.

Discussion

You can use this callback to reschedule the Carbon event loop timer if the next QuickTime task is needed sooner than projected by `QTGetTimeUntilNextTask`. This callback may be called at interrupt time or from another thread on Mac OS X.

See Also

This callback is installed by `QTInstallNextTaskNeededSoonerCallback`. See also `NewQTNextTaskNeededSoonerCallbackUPP`.

TextMediaProc

A callback that can be called whenever a text sample is displayed in a movie.

```
typedef OSErr (*TextMediaProcPtr) (Handle theText, Movie theMovie, short
*displayFlag, long refcon);
```

If you name your function `MyTextMediaProc`, you would declare it this way:

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

Parameters

theText

A handle to the text being displayed.

theMovie

Specifies the movie for this operation.

displayFlag

Undocumented

refcon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

Return Value

See `Error Codes`. Your callback should return `noErr` if there is no error.

See Also

See the `TextMediaSetTextProc` and `NewTextMediaUPP` functions.

TrackTransferProc

Called when a track is forced to draw into a particular graphics world, which may be different from that of the movie.

```
typedef OSErr (*TrackTransferProcPtr) (Track t, long refCon);
```

If you name your function `MyTrackTransferProc`, you would declare it this way:

```
OSErr MyTrackTransferProc (
    Track    t,
    long     refCon );
```

Parameters

t

A track designator.

refCon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

Return Value

See `Error Codes`. Your callback should return `noErr` if there is no error.

See Also

See the `SetTrackGWorld` and `NewTrackTransferUPP` functions.

VdigIntProc

An interrupt callback called by the video digitizer component each time it starts to display a field.

```
typedef void (*VdigIntProcPtr) (long flags, long refcon);
```

If you name your function `MyVdigIntProc`, you would declare it this way:

```
void MyVdigIntProc (
    long    flags,
    long    refcon );
```

Parameters

flags

Undocumented

refcon

A reference constant that the client code supplies to your callback. You can use this reference to point to a data structure containing any information your callback needs.

See Also

See the `VDSetDigitizerUserInterrupt` and `NewVdigIntUPP` functions.

Document Revision History

This table describes the changes to *QuickTime Callbacks Reference*.

Date	Notes
2006-05-23	New document, based on previously published material, that provides API details of QuickTime callbacks.

Document Revision History

Index