Playing Audio Files with QuickTime

Author: Mark Szymczyk
Last Update: September 6, 2005

Games like to provide background music to enhance the gameplay experience. On Mac OS X the easiest way to play background music is store the music in audio files and use QuickTime to play the files. This article shows you how to use QuickTime to play audio files.

Introduction

One of QuickTime's major functions is playing movies. The term movie makes you think of video, but QuickTime movies aren't limited to video files. Audio files can be movies as well. QuickTime can play MP3, AAC (The format the iTunes Music Store uses), AIFF, and Windows WAV files. All you have to do is write code to play a movie, and you'll be able to play any audio file that QuickTime supports.

This article focuses on using QuickTime for Carbon applications on Mac OS X. I assume you have a set of audio files you want to play that you will be including with your application. Some programs may want to use an Open File dialog box to let the user pick a file to play. Implementing Open File dialog boxes is beyond the scope of this article.

Cocoa applications should use the QTKit framework to play QuickTime movies. I hope to cover the QTKit framework in a future article.

Setting Up Your Xcode Project

To play an audio file with QuickTime in a Carbon application, you must add the QuickTime framework and the audio files you want to play to your Xcode project. Choose Project > Add To Project to add files and frameworks to your project. The QuickTime framework should be in the directory System/Library/Frameworks. In your source code files you must add the header file `Movies.h`.

```
#include <QuickTime/Movies.h>
```

Starting and Ending QuickTime

Call the function `EnterMovies()` to start QuickTime. You should call `EnterMovies()` when you initialize your program. When the user quits your program, call `ExitMovies()` to stop QuickTime.

Loading the Audio File

The hardest part of using QuickTime to play audio files is loading the audio file into memory. There are four steps to take to load an audio file on Mac OS X. The first step is to find the main application bundle by calling the function `CFBundleGetMainBundle()`.

```
CFBundleRef gameBundle = CFBundleGetMainBundle();
```

The second step is to find the file you want to play in the bundle by calling the function `CFBundleCopy ResourceURL()`. This function takes four arguments. The first argument is the bundle you retrieved by calling `CFBundleGetMainBundle()`. The second argument is the file's name. The third argument is the file's extension. The final argument is a subdirectory the operating system should search for the file. You can pass NULL as the final argument, which tells the operating system to search the entire bundle for the file.

```
CFBundleRef gameBundle;
CFStringRef filename;
CFStringRef fileExtension;
CFStringRef subdirectory;
CFURLRef fileLocation;

fileLocation = CFBundleCopyResourceURL(gameBundle, filename,
      fileExtension, subdirectory);
```

Suppose you want to load a file named `Music.mp3`. There are two ways you can call `CFBundleCopyResource URL()`. First, you can use `Music` as the file name and `mp3` as the extension. Second, you can use `Music.mp3` as the file name and NULL as the extension. Both methods work. The first method is slightly faster if you have lots of files in your application bundle.

The third step is to set up the audio file so QuickTime can load the file into memory. The easiest way to set up the audio file is to call the function `QTNewDataReferenceFromCFURL()`. This function was introduced in QuickTime 6.4. QuickTime 6.4 shipped with Mac OS X 10.3, which means `QTNewDataReferenceFromCFURL()` works on Mac OS X 10.3 and later. People running Mac OS X 10.2 have to download and install QuickTime 6.4.

`QTNewDataReferenceFromCFURL()` takes four arguments. The first argument is the file location you retrieved by calling `CFBundleCopyResourceURL()`. The second argument is flags, which you should set to 0. The third argument is the data reference that `QTNewDataReferenceFromCFURL()` returns. The fourth argument is the data type of the data reference that `QTNewDataReferenceFromCFURL()` returns.

```
OSErr error;
Handle dataRef;
OSType dataRefType;
CFURLRef fileLocation;

error = QTNewDataReferenceFromCFURL(fileLocation, 0,&dataRef,
      &dataRefType);
```

The data reference `QTNewDataReferenceFromCFURL()` returns is a handle, a pointer to a pointer. Before calling `QTNewDataReferenceFromCFURL()`, you must allocate memory for the handle by calling `NewHandle()`. The size of the handle is AliasHandle.

```
Handle dataRef;
dataRef = NewHandle(sizeof(AliasHandle));
```

The final step is to load the audio file by calling `NewMovieFromDataRef()`. This function takes five arguments. The first argument is the QuickTime movie `NewMovieFromDataRef()` creates. The second argument is flags, which you can set to 0. The third argument is an ID that specifies the resource containing the movie data. You should pass the value `movieInDataForkResID`, which means the movie data is in the file's data fork. The final two arguments are the data reference and data type you created by calling `QTNewDataReferenceFromCFURL()`.

```
OSErr error;
Handle dataRef;
OSType dataRefType;

short fileID = movieInDataForkResID;
short flags = 0;
error = NewMovieFromDataRef(&soundToPlay, flags, &fileID, dataRef,
      dataRefType);
```

Playing the File

To play an audio file, call the function `StartMovie()` and supply the movie you want to play.

```
Movie soundToPlay;
StartMovie(soundToPlay);
```

One line of code is all it takes to play any movie, even movies with video. If you want to play a movie with video, you must provide a destination for QuickTime to show the video. Interface Builder provides movie views to help those of you interested in playing video. I leave playing video as an exercise for you.

A lot of games like to loop sounds (play them repeatedly) to provide continuous background music. To loop a sound you must call the function `IsMovieDone()` to see if the movie finished playing. If `IsMovieDone()` returns a value of true, call the functions `GoToBeginningOfMovie()` and `StartMovie()` to play the sound again.

```
Movie soundToPlay;

if (IsMovieDone(soundToPlay)) {
      GoToBeginningOfMovie(soundToPlay);
      StartMovie(soundToPlay);
}
```

Pausing Playback

When you want to pause a sound you're playing, call the function `StopMovie()` and supply the movie you want to pause. Call `StartMovie()` to resume playing the movie.

```
Movie soundToPlay;
StopMovie(soundToPlay);
```

Giving QuickTime CPU Time

When you play an audio file with QuickTime, you must periodically give QuickTime CPU time to service the movie it's playing. Servicing the movie keeps it playing smoothly. Call the function `MoviesTask()` to give QuickTime CPU time. This function takes two arguments. The first argument is the movie you're playing. The second argument is the amount of time (in milliseconds) to give QuickTime. If you supply a value of 0, QuickTime services every movie once. Unless you're playing a lot of movies, you can normally get away with supplying a value of 0.

```
Movie soundToPlay;
MoviesTask(soundToPlay, 0);
```

You should call `MoviesTask()` as often as you can to keep the movie playing smoothly. In the application that accompanies this article, I use an event timer. The timer calls a function, where I call `MoviesTask()`. Each time the timer fires, `MoviesTask()` gets called, giving QuickTime enough time to keep the movie playing.

Setting a Sound's Volume

To set a sound's volume, call the function `SetMovieVolume()`. This function takes two arguments. The first argument is the movie whose volume you want to set. The second argument is the volume.

```
Movie soundToPlay;
short volume;
SetMovieVolume(soundToPlay, volume);
```

The range of acceptable volumes is –256 to 256. Negative and zero volumes do not play the sound. What is the point of negative volumes? Negative volumes preserve the volume's absolute value, which makes them useful for muting sounds. Multiply the current volume by –1 to mute the sound, and multiply by –1 a second time to restore the sound's volume.

```
Movie soundToPlay;
short currentVolume = GetCurrentVolume(soundToPlay);
SetMovieVolume(currentVolume * -1);
```

Conclusion

Included with this article is an application for you to download. It loops an MP3 file using QuickTime. It comes with full source code you can use to play audio files in your programs. If you have any questions or comments about this article, send them to mark-AT-meandmark-DOT-com.