GETTING STARTED

With FMOD Ex Programmer's API for iPhone/iPad

UPDATED : SEPTEMBER 2012

LEGAL NOTICE

CONTENTS

Contents

Introduction 4
Support Resources 5
  API documentation 5
  Forums 5
  Email 5
  Videos 5
Installation 6
  Libraries 6
Notes on performance 7
Hardware decoding 7
Mixing with the iPod 8
  Force mixing behavior 8
  Duck iPod audio 8
Forcing output to main speaker while recording 9
Determining if the iPod is playing 9
Restoring playback / handling background interruptions 9
Troubleshooting 10
  Pulsating tone is suddenly audible 10

Introduction

Welcome to the FMOD Ex Programmer's API for iPhone/iPad, the quickest and easiest way to get great sound and music into your Apple iPhone and iPad games. This document will show you how to get started implementing FMOD Ex in your game by pointing you in the direction of detailed API documentation and support resources. While the FMOD Ex Programmer's API presents the same interface on all platforms, each platform does have its own unique features and limitations – iPhone/iPad-specific features/limitations will be listed here along with any hints and tips for getting the most out of FMOD Ex on the iPhone/iPad.

Have fun implementing great audio and drop us a line some time,

The FMOD Team
Melbourne, Australia
www.fmod.org

Support Resources

API documentation

Detailed API documentation can be found in the “documentation” directory/folder of your FMOD Ex Programmer’s API installation. This documentation is your main reference for information on FMOD Ex API classes and functions.

Forums

http://www.fmod.org/forum

This should be your first port of call for further FMOD information and questions on implementation. If you have a question related to FMOD, chances are someone else has already asked it. The FMOD forums are free for all FMOD users and are monitored by the FMOD team as well as being home to a strong community of FMOD developers, from student first-timers to top-level professionals working on games that are household names.

Email

support@fmod.org

This is our main technical support line. It’s monitored directly by the FMOD team and we aim to answer all emails within 24 hours. It’s free for all FMOD users and your issues will be addressed directly by the guys who wrote the code. If you can’t find an answer to your problem on the FMOD forums, shoot us an email and we’ll get right onto it.

Videos

http://www.youtube.com/FMODTV

The FMOD YouTube channel contains a growing number of videos of tutorials relating to FMOD and FMOD Designer. This channel is being added to all the time, so be sure to check back regularly.

Installation

FMOD libraries were built using iOS SDK 6.0

Libraries

Link one of these libraries into your project :

- **/api/lib/libfmodex_iphoneos.a** for using all FMOD features on the iPhone device.
- **/api/lib/libfmodex_iphonesimulator.a** for using all FMOD features on the iPhone simulator.
- **/api/lib/libfmodexL_*.a** for a release version with debug logging to help diagnose any problems.

*NOTE: Do not use **/api/lib/libfmodex.dylib.** This is a Mac dynamic library used for some of the provided tools.*

FMOD also requires that you link with the following frameworks:

- **AudioToolbox.framework**
- **CoreAudio.framework**

Notes on performance

For best performance on the iPhone hardware we recommend using one of the lower spec file formats. For cheapest playback of sound effects leave them as straight PCM or encode with a low cost format such as ADPCM. If you require larger compression ratios we recommend using MP2 however there is more runtime decoding expense. High compression formats like Ogg Vorbis are discouraged due to the relatively high expense incurred for decoding. For music streams (highly compressed) we recommend using AAC or MP3 which leverages the hardware decoding chip.

By default all mixing in FMOD operates at 24KHz, so we recommend resampling sound sources to this rate at production time to save realtime resampling cost for sounds that don't adjust their pitch. The default mixing rate is adjustable via **System::setSoftwareFormat**.

To get a detailed view of how expensive (CPU cost) the current DSP network is, connect to the running game with FMOD Profiler. You can enable this with the **FMOD_INIT_ENABLE_PROFILE** flag passed to **System::init**. Be aware that all CPU measurements will be slightly higher due to the cost of running the profiler.

Hardware decoding

Via the AudioQueue codec FMOD supports decoding AAC, ALAC and MP3. At present iOS devices only have support for decoding one sound with hardware at a time (which may be consumed by playing the iPod). At the cost of extra CPU, iOS 3.0 and newer devices have access to software codecs to support more than one sound of these formats. By default FMOD will try to use hardware, failing that software will be used. If the hardware is in use and software is unavailable, FMOD will return **FMOD_ERR_FORMAT** for AAC / ALAC and use FMODs cross-platform MPEG decoder for MP3.

If you want explicit control over whether hardware or software is chosen you can use the **FMOD_AUDIOQUEUE_CODECPOLICY** enumeration provided in **fmodiphone.h**. This is set with the **audioqueuepolicy** member of the **FMOD_CREATESOUNDEXINFO** structure via **System::createSound**. It is important to be aware that **FMOD_HARDWARE** and **FMOD_SOFTWARE** do not govern this behavior, those flags are concerned with channel mixing. All iOS devices are mixed in software hence setting **FMOD_HARDWARE** will simply default back to **FMOD_SOFTWARE**.

When playing MP3s using the AudioQueue codec, seeking is generally slow for the first time each position is visited. If you need fast random access to a file you can create the sound using the **FMOD_TIMEACCURATE** flag. This will scan the file at load time to determine its accurate length, which has the benefit of creating a seek table to aid in seeking. This is a one-time upfront cost for fast seeking vs. paying the cost at runtime for each unique position.

Mixing with the iPod

If you wish to mix the output of FMOD with any currently playing music on the iPod you can pass in an appropriate audio session category via the **sessionCategory** member of **extradriverdata** during **System::init**. Each audio session affects playback behavior in different ways, refer to **fmodiphone.h** for details.

Force mixing behavior

You can force mixing behavior (if the category doesn't allow it) for **FMOD_IPHONE_SESSIONCATEGORY_MEDIAPLAYBACK** and **FMOD_IPHONE_SESSIONCATEGORY_PLAYANDRECORD** by using the **forceMixWithOthers** member of **FMOD_IPONE_EXTRADRIVERDATA**. **forceMixWithOthers** can also be set at runtime via **FMOD_IPhone_MixWithOtherAudio()**. Be aware that if you wish to do recording while playing the iPod you will need to use **FMOD_IPHONE_SESSIONCATEGORY_PLAYANDRECORD** or the iPod will be stopped.

Duck iPod audio

If you wish to duck (lower) the volume of the iPod for a short period so FMOD sound can be clearly heard you can use **FMOD_IPhone_DuckOtherAudio**. When toggling this behavior the FMOD system will re-initialize itself internally so there may be a momentary glitch in playback. This is a side effect of how audio session flags must be applied and is unavoidable at this stage.

Example :

```
FMOD_IPHONE_EXTRADRIVERDATA extradriverdata;

memset(&extradriverdata, 0, sizeof(FMOD_IPHONE_EXTRADRIVERDATA));

extradriverdata.sessionCategory = FMOD_IPHONE_SESSIONCATEGORY_MEDIAPLAYBACK;
extradriverdata.forceMixWithOthers = true;

result = system->init(32, FMOD_INIT_NORMAL, &extradriverdata);
ERRCHECK(result);

// Other code here...

result = FMOD_IPhone_DuckOtherAudio(TRUE);
ERRCHECK(result);

// Other code here...

result = FMOD_IPhone_DuckOtherAudio(FALSE);
ERRCHECK(result);

// Other code here...

result = FMOD_IPhone_MixWithOtherAudio(FALSE);
ERRCHECK(result);
```

Forcing output to main speaker while recording

When recording and playing at the same time, the iPhone defaults to playing output through the receiver speaker (which is used for phone calls). The reason for this is to prevent feedback, however you can choose to override this default. Please refer to **FMOD_IPHONE_EXTRADRIVERDATA** in **fmodiphone.h** for details.

Example :

```
FMOD_IPHONE_EXTRADRIVERDATA extradriverdata;

memset(&extradriverdata, 0, sizeof(FMOD_IPHONE_EXTRADRIVERDATA));

extradriverdata.forceSpeakerOutput = true;

result = system->init(32, FMOD_INIT_NORMAL, &extradriverdata);
ERRCHECK(result);
```

Determining if the iPod is playing

If you need to know whether iPod audio is currently playing (perhaps so you can disable in-game music) you can call the **FMOD_IPhone_OtherAudioIsPlaying** function anytime. Please refer to **FMOD_IPHONE_EXTRADRIVERDATA** in **fmodiphone.h** for details.

Restoring playback / handling background interruptions

When interruptions occur FMOD will generally take care of restarting playback when the interruption has completed, one special exception to this case is when using multitasking. If your application audio has been interrupted while in the background, the OS won't notify when that interruption has gone away. You can overcome this problem by always calling **FMOD_IPhone_RestoreAudioSession()** when the application returns to the foreground i.e. in your app delegate **applicationWillEnterForeground** function.

Troubleshooting

Find solutions for common platform specific issues here:

A quick note. Use the logging version of FMOD to get information in the tty or output log file.

Pulsating tone is suddenly audible

This is a fatal warning from FMOD's mixer. It means the mixer tried to allocate some memory and failed. Because of unexpected behavior at this point, the mixer sends a pulsating sine wave out through the speakers to let you know of this fact.

The solution for this is to reduce memory usage or provide more memory to FMOD, then restart the application.

Note that the tty/log output will display out of memory error messages, and System::setCallback can be used in the API to catch out of memory errors with `FMOD_SYSTEM_CALLBACKTYPE_MEMORYALLOCATIONFAILED`.