GETTING STARTED

With FMOD Ex Programmer's API for Windows

UPDATED : MARCH 2012

LEGAL NOTICE

CONTENTS

Contents

fmod®
Interactive Audio Technology

Introduction

Welcome to the FMOD Ex Programmer's API for Windows, the quickest and easiest way to get great sound and music into your Microsoft Windows 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 - Windows-specific features/limitations will be listed here along with any hints and tips for getting the most out of FMOD Ex on Windows.

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

Which dll to use

Use **api/fmodex.dll** to use the low level 32bit FMOD Ex API. This includes access to the System/Sound/Channel/ChannelGroup/SoundGroup/Geometry and DSP functionality.

Use **api/fmodexL.dll** to add debug logging with FMOD. The log will appear in the executable directory as fmod.log. Control logging with FMOD::Debug_SetLevel.

There are 64bit versions of the DLLs here as well for your convenience.

Which import library to link to to get access to the DLL

Find these libraries in **/api/lib** and put one of them in your linker settings for your project.

Compiler	fmodex.dll	fmodexL.dll	fmodex64.dll	fmodexL64.dll
Visual Studio	fmodex_vc.lib	fmodexL_vc.lib	fmodex64_vc.lib	fmodexL64_vc.lib
Metrowerks Codewarrior	fmodex_vc.lib	fmodexL_vc.lib	fmodex64_vc.lib	fmodexL64_vc.lib
Intel compiler	fmodex_vc.lib	fmodexL_vc.lib	fmodex64_vc.lib	fmodexL64_vc.lib
Borland	fmodex_bc.lib	fmodexL_bc.lib	N/A	N/A
LCC-Win32	fmodex_lcc.lib	fmodexL_lcc.lib	N/A	N/A
Dev-C++, MinGW, Cygwin	libfmodex.a	libfmodexL.a	N/A	N/A

FMOD Event API

Which dll to use

Use **fmoddesignerapi/api/fmod_event.dll** to use the event system 32bit API. This includes access to the EventSystem/EventProject/Event/EventGroup/EventCategory and other functionality.

Use **fmoddesignerapi/api/fmod_eventL.dll** to add debug logging with FMOD Event System. The log will appear in the executable directory as fmod.log. Control logging with FMOD::Debug_SetLevel.

Use **fmoddesignerapi/api/fmod_event_net.dll** to use the event system 32bit API with live update/network tweaking support. This allows FMOD Designer to control the audio settings while the game is running.   Use this as a replacement for **fmod_event.dll** not as an addition.

Use **fmoddesignerapi/api/fmod_event_netL.dll** to add debug logging with the FMOD Event System Network version. The log will appear in the executable directory as fmod.log. Control logging with FMOD::Debug_SetLevel.

There are 64bit versions of the DLLs here as well for your convenience.

Which import library to link to to get access to the DLL

Link the library that matches the dll name in the **fmoddesigner/api/lib** folder. Note that these are Microsoft Visual Studio compatible libraries only. Only link 1 of these libraries and not a combination of them.

Recommended startup sequence (IMPORTANT!)

Due to configuration issues on Windows user's machines, this following code fixes the following issues:

- Speaker configuration in Windows being ignored and just defaulting to stereo. (see use of **System::getDriverCaps** and '**controlpanelspeakermode**' parameter, which is then passed to **System::setSpeakerMode**)
- Stuttering audio due to the user having their 'Hardware acceleration' slider set to 'off' in XP. (see check for **FMOD_CAPS_HARDWARE_EMULATED**, which then increases the FMOD DSP buffersize to over 200ms with **System::setDSPBufferSize**)
- Speaker configuration being set to a setting that the soundcard *doesn't actually support* (See check for **FMOD_ERR_OUTPUT_CREATEBUFFER**, which then triggers a re-initialization with **FMOD_SPEAKERMODE_STEREO**)

Note: The following code must be used for shipping games! Do not ship a game without a startup sequence based on this code!

Use the following code as a basis for your Windows start up sequence:

```
FMOD::System     *system;
FMOD_RESULT       result;
unsigned int      version;
int               numdrivers;
FMOD_SPEAKERMODE speakermode;
FMOD_CAPS         caps;
char              name[256];

/*
    Create a System object and initialize.
*/
result = FMOD::System_Create(&system);
ERRCHECK(result);

result = system->getVersion(&version);
ERRCHECK(result);

if (version < FMOD_VERSION)
{
    printf("Error!  You are using an old version of FMOD %08x.  This program requires %08x\n",
version, FMOD_VERSION);
    return 0;
}

result = system->getNumDrivers(&numdrivers);
ERRCHECK(result);

if (numdrivers == 0)
{
    result = system->setOutput(FMOD_OUTPUTTYPE_NOSOUND);
    ERRCHECK(result);
}
else
{
    result = system->getDriverCaps(0, &caps, 0, 0, &speakermode);
    ERRCHECK(result);

    /*
        Set the user selected speaker mode.
    */
    result = system->setSpeakerMode(speakermode);
    ERRCHECK(result);

    if (caps & FMOD_CAPS_HARDWARE_EMULATED)
    {
        /*
            The user has the 'Acceleration' slider set to off!  This is really bad
            for latency! You might want to warn the user about this.
        */
        result = system->setDSPBufferSize(1024, 10);
        ERRCHECK(result);
    }

    result = system->getDriverInfo(0, name, 256, 0);
    ERRCHECK(result);

    if (strstr(name, "SigmaTel"))
    {
        /*
            Sigmatel sound devices crackle for some reason if the format is PCM 16bit.
            PCM floating point output seems to solve it.
        */
        result = system->setSoftwareFormat(48000, FMOD_SOUND_FORMAT_PCMFLOAT, 0,0,
FMOD_DSP_RESAMPLER_LINEAR);
        ERRCHECK(result);
    }
}

result = system->init(100, FMOD_INIT_NORMAL, 0);
if (result == FMOD_ERR_OUTPUT_CREATEBUFFER)
{
    /*
        Ok, the speaker mode selected isn't supported by this soundcard.  Switch it
        back to stereo...
    */
    result = system->setSpeakerMode(FMOD_SPEAKERMODE_STEREO);
    ERRCHECK(result);

    /*
        ... and re-init.
    */
    result = system->init(100, FMOD_INIT_NORMAL, 0);
}
ERRCHECK(result);
```

Important issue with certain compilers and FMOD’s C++ interface

Due to incompatible linking standards with C++ symbols in libraries across different compilers, you will not be able to use the C++ interface of FMOD Ex with the following compilers:

- Borland
- LCC-Win32
- Dev-C++
- MinGW
- Cygwin

You can only use the FMOD Ex C interface with these compilers, as at least that has a compatible standard (i.e. stdcall symbols are always the same format). Each C++ compiler generates its own version of mangled symbols, and the above-mentioned compilers are not compatible with the symbols that MSVC produces, which is what FMOD is compiled in, and is the most popular compiler for commercial development at this stage.

Note that the Intel compiler and Codewarrior do not have this problem, they can resolve MSVC style symbols.

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`.

Stuttering/skipping sound

This is most common when using software mixed sounds or when streaming sounds from disk. Commonly known as buffer underrun/overrun, this problem can be caused by one or more factors:

Bad soundcard drivers

This may be solved by upgrading your soundcard drivers. (Note: it is recommended you get the latest drivers anyway).

CPU issues

Machine too slow, or whatever you are trying to do with FMOD is too CPU intensive! (i.e. playing 100 MP3s at once will most likely bring FMOD to its knees on older hardware, or maybe a user stream callback or DSP callback is spending too much time executing).

Mixer buffersize is set too small

You can increase stability to combat these issues, by increasing FMOD's internal mixing buffer size. This will lead to greater stability but also larger latency on issuing commands to hearing the result. Call **System::setDSPBufferSize** to alter this. See documentation for **System::setDSPBufferSize** for more information.

Stream buffersize is set too small

If you are using the FMOD Ex streamer, you might be streaming from a slow media, such as CDROM or over network, or even a fragmented harddisk, therefore FMOD needs more time to fill its streaming buffer before it runs out. See **System::setStreamBufferSize** to adjust the file read buffer size for the streamer. If the stream is starving because the codec is an expensive codec (and the file media is not to blame) then the problem could be the FMOD stream decode buffer size. You can adjust this using the '**decodebuffersize**' member of the **FMOD_CREATESOUNDEXINFO** structure.

Output type

FMOD_OUTPUTTYPE_DSOUND will provide more solid output than **FMOD_OUTPUTTYPE_WINMM** in anything except Windows NT. This is a problem with Windows Multimedia Services not being as realtime as it should be. Under NT, **FMOD_OUTPUTTYPE_WINMM** is more stable, as DirectSound in NT is just emulated by using WINMM itself and is actually slower and has longer latency! **Note!** Please don't feel the need to use **System::setOutput** if you don't need to. FMOD auto-detects the best output mode based on the operating system.