Lite-C FMOD wrapper DLL v 0.1

for the A7 game engine (Lite-C)

REFERENCE

Table of contents

Last changed: *2007-05-10*

Introduction

Welcome to the reference of the Lite-C FMOD wrapper DLL for the A7 game engine. In the following you will find an introduction to FMOD, a list of differences to the audio system of the A7 engine, an insight into the workflow of the wrapper and an overview of the implemented instructions.

What is „FMOD“?

FMOD is a cross-platform audio engine, which makes it easy for you to use current audio technologies in your software. No other audio engine offers such a rich and up-to-date platform support. FMOD isn't only available for the Windows 32 bit system, but also for 11 other systems, among them e.g. Windows 64bit (AMD64), Linux, Macintosh and all game consoles available on the market. FMOD is being and was used by over one hundred commercial games, for example in "World of Warcraft" or "Flat Out".

Why should I choose an 3rd party audioengine?

The audio abilities of the A7 engine are quite limited. Streaming a file from HD or CD/DVD will produce a remarkable lag and the whole (stream-) initializing freezes the engine noticeably. This is an absolute disaster for professional game productions. Apart from these things you are also limited on how to control the stream (no seeking!) and how to retrieve information about file tags, for instance. You can't assign runtime special effects and features like spectral analysis to enable beat detection et al. With FMOD, all of this will be enabled for you – no matter which edition of the A7 engine you own or if you „just“ use the free Lite-C distribution.

The difference between the FMOD.dll and the Lite-C wrapper-plugin DLL

The FMOD audio engine manages certain object structures, which are not compatible to the procedural ANSI C standard. In order to be able to work with the FMOD functions, you would have to manage these objects yourself. Since this is not possible in Lite-C, the wrapper DLL manages all created objects (Streams, Samples, Channels, DSP unit, etc..) and assigns an unique identification number (as Integer) to them, which will be returned instead of a Stream object or such (that is similar to the handle which will be returned by the media_* instruction of the A7 engine). By these IDs you can comfortably access these objects. I advice you to have both references (this and the official FMOD reference) opened at the same time – just because this reference doesnt explain how FMOD works and what the constants mean, etc.pp. In addition it could also happen that some FMOD function signatures (return type, parameter list) will be different in the wrapper, some functions are missing, there are some special requirements or restrictions or whatever. So its just „good“ to work with both references simultaneously.

Instruction set modifications

You cannot use the FMOD instructions directly, this will be done by the wrapper. You can achieve this by using appropriate wrapper functions. These will receive the parameters, process them maybe and then pass these to the FMOD.dll. The Lite-C wrapper functions all begin with a „LC_“ placed in front of the original FMOD instruction name. Example: If you want to use the FMOD instruction FSOUND_Stream_Open() to open a stream, you would use **LC**_FSOUND_Stream_Open() in Lite-C instead.

The most important innovation is the self-managed FMOD objects. Streams, Samples and (own) DSP unit are administered and managed by the wrapper DLL. You will always receive an unique integer number (as ID) instead of these fancy FMOD object structures. If a FMOD isntruction requires you to pass such an object, you pass the integer instead. The wrapper itself will automatically pick the corresponding object and proceed it. Read in addition the corresponding signatures of the function(s) to notice these requirements!

Constants

Instead of the instruction set, the FMOD constants remain the same. You can use them without any affixes. Use them like it is shown in the FMOD reference. I defined all of them in the plugin header file. Example: You want to play a CD track in loop. You would do this: LC_FSOUND_CD_SetPlayMode (0, FSOUND_CD_PLAYLOOPED); In this case the constant FSOUND_CD_PLAYLOOPED would represent the same FMOD constant, which tells FMOD to always play the tracks of an Audio-CD in loop.

Initializing & Closing FMOD

In contrast to the FMOD engine, which would require you to initialize FMOD manually, you don't need to think about this anymore. The FMOD wrapper will do this automatically at the time of the first FMOD instruction in your Lite-C program. However, you can also initialize the FMOD manually with the appropriate FMOD initialization calls. The wrapper recognizes this and skips the initialization check. When you shut down the game engine, FMOD neds to be terminated as well. You have to do this manually – more information about this in the chapter "Integration of FMOD into your project".

Supported APIs and FMOD features

At this time the Lite-C FMOD wrapper supports only a subset of the complete instruction set of the FMOD audioengine. So far only the FSOUND API is being implemented. The following core features are available:

- Initialization, shutdown and settings
- Samples
- Streams (from HD, disc and internet sources)
- Channels
- CD
- Frequency analysis

These features are almost completely implemented. Please compare each feature to the original FMOD instruction. The following features are not yet implemented and/or are still in development. I hope, you have an understanding for this. This concerns the following features without a specific order of priorities:

- 3D sound, multiple listener objects, surround sound
- runtime channel effects
- samples, which are loaded from WRS resources
- sound recording (i.e. with a mic)
- syncpoints and callbacks in musicfiles; internal system-callbacks
- FMUSIC API

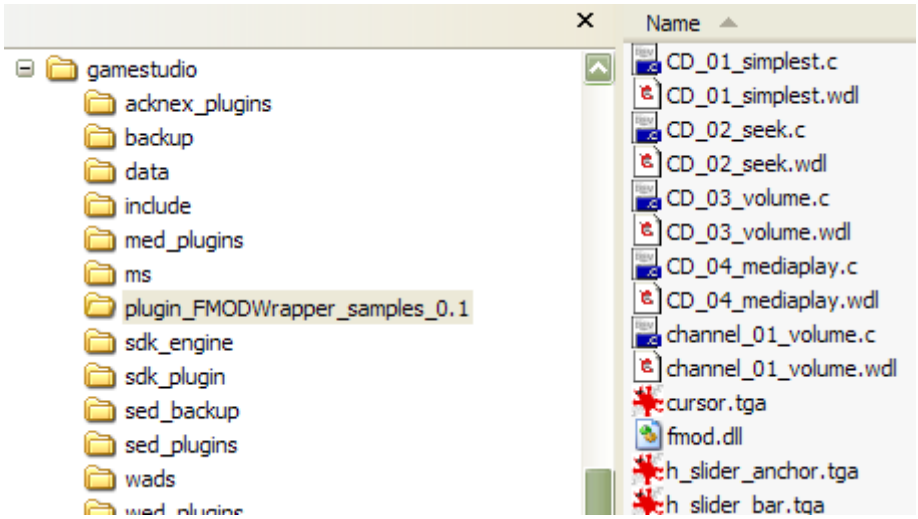
Requirements

The FMOD wrapper supports only Lite-C. If you want to work with the wrapper, you have to fulfill the following criteria:

- You own the A6 engine and you are a betatester: by this you have access to the latest Lite-C beta and able to use the wrapper (though, only with the beta and not with an A6 release version).
- You own an A7 game engine license and by this also a valid copy of Lite-C.
- You don't own a copy of 3D Gamestudio. Then you could download the free version of Lite-C on the Conitec downloads-page: http://www.conitec.com
- You have to download the FMOD 3.75 DLL on your own. The FMOD DLL is NOT being included in this package! You have to download the DLL, if you did'nt done this alrady! The DLL is freely available on the FMOD webpage: http://www.fmod.de

Installation of the examples

Theoretically you can extract the archive with the example files everywhere. However, if you want to use the provided RUN_*.bat files, to watch/use/test out the examples directly, then you have to extract the archive simply into your Gamestudio or Lite C folder. For instance:



Then, if you start the *.bat files, the corresponding example will be opened. Make sure that you copied the wrapper AND the FMOD.dll into the example folder (the wrapper and the FMOD.dll are not contained in the examples archive!).

Integration of FMOD into your project

The installation of FMOD and the wrappers is really easy. In the following step by step guide you will learn how to integrate the FMOD audio engine into your project:

1. Copy the files fmod.dll, LC_FMOD_WRAP.dll and LC_FMOD_WRAP.h into your project's folder.
2. Include the plugin headerfile: #include "LC_FMOD_WRAP.h"; You have to include it befor you make a single FMOD call! Place it simply right after you included the acknex.h and default.c files.

```
1 //- includes ------------------------------------------------
2
3     #include <acknex.h>
4     #include <default.c>
5
6     #include "LC_FMOD_WRAP.h";  //FMOD plugin header file
```

3. When the game engine shuts down, you have to shut down FMOD as well – otherwise you will get a memory leak! You can resolve this with different approaches: select the method which suits you most.

- Shut down the FMOD engine manually, before you execute sys_exit(); :

```
#ifdef FMOD_DLL
	LC_FSOUND_Close();
#endif
```

- You could also let the game engine shut down FMOD automatically by using the on_exit event:
    - If you don't use the on_exit event, you can simply write a new void function, which shuts down FMOD and is being assigned to the on_exit event:

```
void sysExit_event ()
{
   #ifdef FMOD_DLL
      LC_FSOUND_Close();
   #endif
}


void main ()
{
   //...
   on_exit = sysExit_event;
}
```

    - If you already own an on_exit event, just copy and paste the above code into that function!
    - Optionally, you can also assign directly the FMOD shutdown function to the on_exit event: `on_exit = LC_FSOUND_Close();`

You don't have to care about the initialization: the FMOD engine will be booted *automatically* at the time when you first call a FMOD instruction. I**t is just important that you shut down the FMOD engine properly!**

By this, FMOD and the wrapper are successfully integrated into your project. If you are new to FMOD and advanced audio programming, I advice you to head to the tutorials first and the example files to have a rough overview how everything works together.

Basic explanations and the corresponding examples

In the following all important function groups are brought up with a simple explanation and corresponding referrals to the examples. For further information, read through the FMOD reference.

Streams

All musicfiles which are large and/or played very rarely are commonly played as so called streams. A stream is an active „connection“ between the file and the program. While the musicfile is being

played, a small part is always loaded into the memory and the used parts („chunks“) are being removed. By this, the overall memory consumption is much smaller as if you would load the whole file into the memory - this is particularly interesting for long musicfiles (songs, ambient sounds, speaker comments/voice overs, etc.). With FMOD you can also open streams to files on disc-media (CD, DVD) and files on the internet.

stream_01_simplest.c	This shows the simplest way of how to open a stream and play it. Furthermore, it introduces the way you can achieve the automatic shutdown of FMOD.
stream_02_seek.c	Demonstration of FMOD's capability to seek in a running stream.
stream_03_stereo.c	This sample plays a stream and visualizes the runtime-volume (also called "level(s)") of the stereo channels (left and right speaker, for instance).
stream_04_mediaplay.c	The mediaplay-demo demonstrates the basic instructions to control a stream: pause, stop and play.
stream_05_information.c	In this application you retrieve some static and runtime-affected information of the stream (tags not included).
stream_06_tags.c	One prominent feature of mediaplayers is to show artist and title information which are being saved inside the music file. This example tells you how to do it.
stream_07_spectrum.c	Displaying the spectrum of a song is one of the coolest features of each mediaplayer or Hifi-stations. With FMOD it is very easy have access to these spectrum values!

Samples

Samples are music files, which are completely loaded into memory (they are being extracted, if they are compressed, like *.ogg files). Due to the high memory consumption, sample allocation is only meaningful, if the file is short and being played frequently (i.e. For typical soundeffects such as gunshots, explosions, etc.).

sample_01_simplest.c	This example loads a sample and plays it when you press a button.
sample_02_loop.c	At program start a sample is being loaded and played in loop.

CD

With FMOD you can also play tracks form an audio CD (and/or the audio sector of a hybrid CD). The FMOD engine uses its own routine in order to play audio CDs. Contrary to conventional mediaplayers (e.g. the Windows Media Player), the CD isn't being queried at any time for its status – which would produce lags, lower framerates, etc. While a song is being played, FMOD never queries the drive, which leads to a 0% CPU usage - compared to conventional solutions this is absolutely superior!

CD_01_simplest.c	This demo plays simply the first track of the primary drive.
CD_02_seek.c	You are also enbaled to seek on CD tracks! This demo shows it.
CD_03_volume.c	This applications demonstrates how to change the CD volume.

CD_04_mediaplay.c	This mediaplayer demo is a bit more advanced as the stream-counterpart: You are also able to skip to the next or previous CD track.

Channels

Instead of dealing with individual streams, samples etc., you can also divide these into so called channels, "tracks" or you can also say: groups. Then, you can mix channels differently than other ones (different volumes, added effects, etc.). Example: If the user-avatar in a game is wounded, you could lower the volume of the ambient channel and increase the player related noises rapidly (play a slight echo effect, for instance) in order to simulate a more immersive game feeling (e.g. in Call of Duty 2). This could be done by make one channel louder and the second more quietly. You don't have information about which streams or samples are being played – you just focus in the channels. You just have to take care during sample/stream creation that you start playing these in the corresponding channels!

channel_01_volume.c	This cool demo makes you able to change various volume related things on a channel like volume, pan and frequency!

System

system_01_information.c	This program shows you some important internal information values which might be interesting for you.

Overview of the FSOUND API implementation

In the following the instruction set of the wrapper DLL will be specified. The categories are sorted according to the FMOD reference, so that you can easily compare original and wrapper instructions. The instructions are coloured in order to emphasize whether they are implemented or not, and/or if they have other characteristics. **The overview is'nt going to deal with the specific function signatures (return value and parameter list)!** - please compare yourself the signatures of the implemented functions with the appropriate FMOD function by using this reference, the plugin header file and the official FMOD reference file.

key:		
	function	supported function
	~~function~~	unsupported function, which will be implemented in a future release
	~~function~~	substituted function
	function	replacement function
	function	additional function, which just exists in the wrapper DLL, but not in FMOD
	~~*function*~~	for technical reasons this function won't be implemented at any time

Pre Initialization / Initialization / Enumeration Functions

LC_FSOUND_Close
~~*LC_FSOUND_File_SetCallbacks*~~
LC_FSOUND_Init
LC_FSOUND_SetBufferSize
LC_FSOUND_SetDriver
LC_FSOUND_SetHWND
LC_FSOUND_SetMaxHardwareChannels
~~*LC_FSOUND_SetMemorySystem*~~
LC_FSOUND_SetMinHardwareChannels
LC_FSOUND_SetMixer
LC_FSOUND_SetOutput

Global Run-Time Update Functions

LC_FSOUND_SetPanSeperation
LC_FSOUND_SetSFXMasterVolume
LC_FSOUND_SetSpeakerMode
LC_FSOUND_Update

Global Run-Time Information Functions

LC_FSOUND_GetCPUUsage
LC_FSOUND_GetChannelsPlaying
LC_FSOUND_GetDriver
~~LC_FSOUND_GetDriverCaps~~
instead of this function, please use these functions:
LC_FSOUND_GetDriverCaps_HARDWARE

LC_FSOUND_GetDriverCaps_EAX2
LC_FSOUND_GetDriverCaps_EAX3
LC_FSOUND_GetDriverName
LC_FSOUND_GetError
LC_FSOUND_GetMaxSamples
LC_FSOUND_GetMaxChannels
LC_FSOUND_GetMemoryStats
LC_FSOUND_GetNumDrivers
LC_FSOUND_GetNumHWChannels
alternatively you can also use these functions:
LC_FSOUND_GetNumHWChannels_num2d
LC_FSOUND_GetNumHWChannels_num3d
LC_FSOUND_GetNumHWChannels_total
LC_FSOUND_GetOutput
LC_FSOUND_GetOutputHandle
LC_FSOUND_GetOutputRate
LC_FSOUND_GetSFXMasterVolume
LC_FSOUND_GetVersion

Sample Functions

LC_FSOUND_Sample_Alloc
LC_FSOUND_Sample_Free
~~*LC_FSOUND_Sample_Get*~~
LC_FSOUND_Sample_GetDefaults
LC_FSOUND_Sample_GetDefaultsEx
LC_FSOUND_Sample_GetLength
LC_FSOUND_Sample_GetLoopPoints
LC_FSOUND_Sample_GetMinMaxDistance
LC_FSOUND_Sample_GetMode
LC_FSOUND_Sample_GetName
LC_FSOUND_Sample_Load
(Notice: `int index` is supported by the function, but at this time it will be automatically replaced with `FSOUND_FREE`)
~~LC_FSOUND_Sample_Lock~~
LC_FSOUND_Sample_SetDefaults
LC_FSOUND_Sample_SetDefaultsEx
LC_FSOUND_Sample_SetMaxPlaybacks
LC_FSOUND_Sample_SetMinMaxDistance
LC_FSOUND_Sample_SetMode
~~LC_FSOUND_Sample_SetLoopPoints~~
~~LC_FSOUND_Sample_Unlock~~
~~LC_FSOUND_Sample_Upload~~

Channel Functions

LC_FSOUND_PlaySound
~~LC_FSOUND_PlaySoundEx~~
LC_FSOUND_StopSound
LC_FSOUND_SetFrequency
LC_FSOUND_SetLoopMode
LC_FSOUND_SetMute
LC_FSOUND_SetPan
LC_FSOUND_SetPaused

LC_FSOUND_SetPriority
LC_FSOUND_SetReserved
LC_FSOUND_SetSurround
LC_FSOUND_SetPriority
LC_FSOUND_SetVolume
LC_FSOUND_SetVolumeAbsolute
LC_FSOUND_SetVolumeAbsolute_global
LC_FSOUND_GetVolume
LC_FSOUND_GetAmplitude
~~LC_FSOUND_3D_SetAttributes~~
~~LC_FSOUND_3D_SetMinMaxDistance~~
LC_FSOUND_SetCurrentPosition
LC_FSOUND_GetCurrentPosition
LC_FSOUND_GetCurrentSample
LC_FSOUND_GetCurrentLevels
LC_FSOUND_GetFrequency
LC_FSOUND_GetLoopMode
LC_FSOUND_GetMixer
LC_FSOUND_GetMute
LC_FSOUND_GetNumSubChannels
LC_FSOUND_GetPan
LC_FSOUND_GetPaused
LC_FSOUND_GetPriority
LC_FSOUND_GetReserved
LC_FSOUND_GetSubChannel
LC_FSOUND_GetSurround
LC_FSOUND_IsPlaying
~~LC_FSOUND_3D_GetAttributes~~
~~LC_FSOUND_3D_GetMinMaxDistance~~

3D Sound Functions

~~LC_FSOUND_3D_Listener_GetAttributes~~
~~LC_FSOUND_3D_Listener_SetAttributes~~
~~LC_FSOUND_3D_Listener_SetCurrent~~
~~LC_FSOUND_3D_SetDistanceFactor~~
~~LC_FSOUND_3D_SetDopplerFactor~~
~~LC_FSOUND_3D_SetRolloffFactor~~

Stream Functions

~~LC_FSOUND_Stream_AddSyncPoint~~
LC_FSOUND_Stream_Close
~~LC_FSOUND_Stream_Create~~
~~LC_FSOUND_Stream_CreateDSP~~
~~LC_FSOUND_Stream_DeleteSyncPoint~~
LC_FSOUND_Stream_FindTagField
LC_FSOUND_Stream_GetLength
LC_FSOUND_Stream_GetLengthMs
LC_FSOUND_Stream_GetLengthSecs
LC_FSOUND_Stream_GetMode
LC_FSOUND_Stream_GetNumSubStreams
LC_FSOUND_Stream_GetNumSyncPoints
LC_FSOUND_Stream_GetNumTagFields

LC_FSOUND_Stream_GetOpenState
LC_FSOUND_Stream_GetPosition
~~LC_FSOUND_Stream_GetSample~~
~~LC_FSOUND_Stream_GetSyncPoint~~
~~LC_FSOUND_Stream_GetSyncPointInfo~~
LC_FSOUND_Stream_GetTagField
LC_FSOUND_Stream_GetTime
LC_FSOUND_Stream_GetTime_secs
~~LC_FSOUND_Stream_Net_GetBufferProperties~~
LC_FSOUND_Stream_Net_GetLastServerStatus
LC_FSOUND_Stream_Net_GetStatus
LC_FSOUND_Stream_Net_SetBufferProperties
~~LC_FSOUND_Stream_Net_SetMetadataCallback~~
LC_FSOUND_Stream_Net_SetProxy
LC_FSOUND_Stream_Open
LC_FSOUND_Stream_Play
~~LC_FSOUND_Stream_PlayEx~~
LC_FSOUND_Stream_SetBufferSize
~~LC_FSOUND_Stream_SetEndCallback~~
LC_FSOUND_Stream_SetLoopCount
~~LC_FSOUND_Stream_SetLoopPoints~~
LC_FSOUND_Stream_SetMode
LC_FSOUND_Stream_SetPosition
LC_FSOUND_Stream_SetSubStream
~~LC_FSOUND_Stream_SetSubStreamSentence~~
~~LC_FSOUND_Stream_SetSyncCallback~~
LC_FSOUND_Stream_SetTime
LC_FSOUND_Stream_SetTime_secs
LC_FSOUND_Stream_Stop

CD Functions

LC_FSOUND_CD_GetNumTracks
LC_FSOUND_CD_GetPaused
LC_FSOUND_CD_GetTrack
LC_FSOUND_CD_GetTrackLength
LC_FSOUND_CD_GetTrackLength_secs
LC_FSOUND_CD_GetTrackTime
LC_FSOUND_CD_GetTrackTime_secs
LC_FSOUND_CD_GetVolume
LC_FSOUND_CD_OpenTray
LC_FSOUND_CD_Play
LC_FSOUND_CD_SetPaused
LC_FSOUND_CD_SetPlayMode
LC_FSOUND_CD_SetTrackTime
LC_FSOUND_CD_SetVolume
LC_FSOUND_CD_Stop

DSP Functions

LC_FSOUND_DSP_ClearMixBuffer
~~LC_FSOUND_DSP_Create~~
LC_FSOUND_DSP_Free
LC_FSOUND_DSP_SetActive

LC_FSOUND_DSP_GetActive
LC_FSOUND_DSP_GetBufferLength
LC_FSOUND_DSP_GetBufferLengthTotal
LC_FSOUND_DSP_SetPriority
LC_FSOUND_DSP_GetPriority
~~LC_FSOUND_DSP_GetClearUnit~~
~~LC_FSOUND_DSP_GetClipAndCopyUnit~~
~~LC_FSOUND_DSP_GetMusicUnit~~
~~LC_FSOUND_DSP_GetSFXUnit~~
~~LC_FSOUND_DSP_GetFFTUnit~~
instead of this function, please use these functions:
LC_FSOUND_DSP_OpenFFTUnit
LC_FSOUND_DSP_CloseFFTUnit
LC_FSOUND_DSP_GetSpectrum
~~LC_FSOUND_DSP_MixBuffers~~

FX Functions

~~LC_FSOUND_FX_Disable~~
~~LC_FSOUND_FX_Enable~~
~~LC_FSOUND_FX_SetChorus~~
~~LC_FSOUND_FX_SetCompressor~~
~~LC_FSOUND_FX_SetDistortion~~
~~LC_FSOUND_FX_SetEcho~~
~~LC_FSOUND_FX_SetRanger~~
~~LC_FSOUND_FX_SetGargle~~
~~LC_FSOUND_FX_SetI3DL2Reverb~~
~~LC_FSOUND_FX_SetParamEQ~~
~~LC_FSOUND_FX_SetWavesReverb~~

Recording Functions

~~LC_FSOUND_Reverb_SetProperties~~
~~LC_FSOUND_Reverb_GetProperties~~
~~LC_FSOUND_Reverb_SetChannelProperties~~
~~LC_FSOUND_Reverb_GetChannelProperties~~

FAQ

- **Q:** The Wrapper works only in connection with the A7 engine. Is there also a version available for the A6 game engine in correlation with the C-Script scripting language? - **A:** The Wrapper works only with Lite C. A port to C-Script is not planned at the present time, because the development of the Lite-C version is not even finished. If you should have an urgent need for a port of a certain subset of the current supported instruction set (e.g. streaming), then please contact me.

- **Q:** Can I use the Lite-C version of the wrapper in my A6 project? - **A:** Yes and no. If you are a betatester, you have full access to the current Lite-C beta which works also with the

latest A6 beta release, so that you can indeed use the Wrapper with A6. You can also bind the A6 engine into a C++ program, but this makes no sense, because then you could use instantly the „real“ FMOD audioengine instead of the wrapper.

Debugging

- **Q:** In the moment I execute my very first FMOD instruction, the game engine chrashes. Could it be the case, that the wrapper DLL is broken? - **A:** For sure, this could be the case. But in this special case, it is obvious, that the game engine does'nt find the FMOD.dll and/or the wrapper DLL. This is cause by a non-existent or wrong plugindir declaration. Have a look into the example folder: each .c file has an own .wdl file which says he engine, that the plugin dlls are being found in the main game folder. Check your own plugindir on that basis and retry!

- **Q:** When I quit the game, I get everytime a runtime error message about a memory leak or such.. what happens there? - **A:** Probably you forgot to shut the FMOD engin down. Add the appropriate instruction into your program code (see: Integration into your project).

- **Q:** I shut FMOD already down, when I terminate the game engine. Nevertheless, I still get this error message. - **A:** Make sure that you don't make a call to the FMOD wrapper after you have shut it down. If you have still problems, please contact me.

License and terms of use

It is at the express consent of the parties that the present agreement be written in English. C'est à la demande expresse des parties que cette convention soit rédigée en anglais.

READ THE TERMS OF THIS AGREEMENT ("AGREEMENT") CAREFULLY BEFORE USING THE SOFTWARE PACKAGE.

The subject of this license agreement is the computer program "Lite-C FMOD wrapper DLL", the program description and the instruction manual, as well as other associated written and electronic materials, in the following termed as software. Christian Behrenberg wants to point out the fact that it is not possible at the state of the art to provide computersoftware in such a way, that it works in all applications and combinations error free. The subject of this contract is therefore only a software, which is in the sense of the programs description and the instruction manual principly useful. BY USING THIS SOFTWARE, YOU AGREE TO THE TERMS OF THIS AGREEMENT.

Christian Behrenberg is both author and owner of the software, as well of in this software used algorithms and procedures. Christian Behrenberg keeps the legal claim and tenure at the software. You accept that this granted license is not a sale of the software and that this license does'nt entitle you to assert a claim on patents, duplication, industry secrets, registered trademarks or on other rights.

You are allowed to copy, distribute and use the software freely (freeware principle), as far as the following regulations are considered: The software has to be distributed in the EXE- or ZIP archives which are being provided by Christian Behrenberg and you may not alter or add a file(s) in the distributed contents. The licensee may use the software in commercial products. If the licensee uses the software and thereby publishes a product with it, you have to inform Christian Behrenberg and leave him a free copy of that product. Furthermore the licensee has to add Christian Behrenberg as developer to his documentation of the product specified as "additional audio engineering", "plugin programming", or as "freelance audio engineer", "freelance audio programmer" or "freelance game developer". Optionally the licensee is permitted to print a reference to the internet-website of Christian Behrenberg in the documentation of the product (http://www.christian behrenberg.de). The licensee is bound to the license agreements of all external software, which he uses automatically by the use of this software. Christian Behrenberg is not responsible for license injuries of third parties by the inappropriate use of this software. The licensee accepts all license agreements of external software with this license agreement. This concerns the "FMOD audio engine" in this case, copyright © 2001-2002 Firelight Technologies, Pty, Ltd. All rights reserved.

YOU MAY NOT, AND YOU MAY NOT PERMIT OTHERS TO REVERSE ENGINEER, DECOMPILE, DECODE, DECRYPT, DISASSEMBLE, OR IN ANY WAY DERIVE SOURCE CODE FROM THE SOFTWARE; YOU MAY NOT, AND YOU MAY NOT PERMIT OTHERS TO SELL, RENT, LEASE OR OTHERWISE EXPLOIT THE SOFTWARE.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL CHRISTIAN BEHRENBERG BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR SPECIAL , INDIRECT, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF OR RELATE D TO THE USE OF OR INABILITY TO USE SOFTWARE, EVEN IF CHRISTIAN BEHRENBERG HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

THE SOFTWARE AND ACCOMPANYING WRITTEN MATERIALS ARE PROVIDED ON AN "AS IS" BASIS, WITHOUT ANY WARRANTIES OF ANY KIND, INCLUDING, BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. NO ORAL OR WRITTEN INFORMATION OR ADVICE GIVEN BY CHRISTIAN BEHRENBER SHALL CREATE A WARRANTY, OR IN ANY WAY INCREASE THE SCOPE OF THIS WARRANTY, AND YOU MAY NOT RELY ON ANY SUCH INFORMATION OR ADVICE. CHRISTIAN BEHRENBERG DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE OR THE RESULTS OF USE, OF THE SOFTWARE OR WRITTEN MATERIALS IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, CURRENTNESS, OR OTHERWISE, AND THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE SOFTWARE IS ASSUMED BY YOU. IF THE SOFTWARE OR WRITTEN MATERIALS ARE DEFECTIVE, YOU, AND NOT CHRISTIAN BEHRENBERG ASSUME THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION OTHER THAN EXPRESSLY DESCRIBED ABOVE.

Any action related to this Agreement will be governed by German law. No choice of law rules of any jurisdiction will apply. You acknowledge that you have read this Agreement, understand it, and agree to be bound by its terms and conditions.

Development information

The Lite-C FMOD wrapper DLL is being developed by Christian Behrenberg. You'll find further information on his website http://www.christian-behrenberg.de for informationen, news, author's corrections, announcements and new releases.