**术语/基本概念**

**Samples,bytes and milliseconds**

一个左声道+右声道叫做一个sample

如果是16位立体声，1 sample = 16bits = 4bytes

如果该音频是44.1khz，那么就是每秒钟有44100个sample，那么一个sample就是1/44100秒。

一段音频，其长度可以用不同的单位表示，如毫秒，比特或者samples，函数Sound::getLength可以根据你需要的单位来得到长度。

**Samples,Compressed samples and streams**

当声音被加载时，它要么作为静态样本作为PCM（samples）被解压缩到内存中，要么以其本机格式加载到内存中并在运行时被解压缩（compressed samples），要么从外部媒体（如硬盘或CD（streams））实时流式传输和解码（以块为单位）。**简单来说，samples在内存中是已经被解压过的格式，compressed samples是还未被解压的格式，那么运行时要实时解压，streams就是从外部媒体实时传输和解码（不经过内存？）。**

Samples:适用于需要一次播放多次的小声音，例如声音效果。它们通常只需要很少或根本不需要CPU来回放，并且可以硬件加速。

Streams:“Streams”适用于太大而无法放入内存，需要从磁盘流到FMOD管理的小环形缓冲区的大声音。根据文件格式，它们占用少量的CPU和磁盘带宽。例如，MP3比PCM解压缩的wav文件需要更多的cpu能量来实时解码。流媒体声音一次只能播放一次，不能像一个示例一样一次多次生成，因为每个流只有一个文件句柄和一个要解码的环形缓冲区。

Compressed samples:“Compressed samples”是一个选项，允许用户加载特定的压缩文件格式（例如当前的IMA ADPCM、FADPMCM、Vorbis、MP2、MP3、AT9和XMA格式）。FADPCM/Vorbis/AT9仅通过.FSB容器格式受支持，并将它们压缩在内存中而不解压缩。它们是混合在CPU上的软件，没有流的“仅一次”限制。它们比标准的PCM样本占用更多的cpu，但实际上由于没有进行任何磁盘访问和更小的内存缓冲区，因此比流占用更少的cpu。

**System::createSound函数默认会去解压整个音频放入内存（就是samples）**

**Channels and sounds**

当你load声音后，可以调用System::playSound来播放它，这个函数会返回一个指向Channel的指针。

FMOD将会自动选择一个channel来用作声音的播放，你不需要去管理channels。

**Sub-mixing and ChannelGroups**

如果你不想对每个单独的channel处理或控制，channel可以放置在ChannelGroup中。ChannelGroup允许你对一组channels进行同一操作，并控制音频的混合图。

ChannelGroup也可以被认为是“sub mix”，就像在信号链中一样，这将是信道信号混合到单个缓冲器中的点。然后，可以使用DSP效果（见下文）对该缓冲区进行一次处理，而不是对每个通道进行一次处理，从而节省大量的CPU时间。

2D and 3D

三维声源是一个在空间中具有位置和速度的channel。当一个3Dchannel正在播放时，它的音量、扬声器的位置和音调将根据与听者的关系自动受到影响。

听众可以是玩家，或者是游戏摄像机。它有一个像声源一样的位置和速度，但它也有一个方向。

听者和声源之间的距离决定了音量。

听者和声源的相对速度决定了音高（多普勒效应）。

听众对声源的方向决定了泛音或说话人的位置。

2D声音的不同之处在于它不受3D声音侦听器的影响，也不受多普勒、衰减或扬声器位置的影响。

二维声音可以调用Channel:：setMixLevelsOutput、Channel:：setMixMatrix或Channel:：setPan，而三维声音上的这些命令没有任何效果，除非调用Channel:：set3DLevel来更改3D声音的2D分量。您可以使用此功能在3D和2D之间混合声音。

3D声音可以调用函数名中包含单词3D的任何函数，而2D声音则不能。

DSP

DSP代表“数字信号处理”，通常与处理原始PCM样本以改变声音有关。FMOD提供了一套DSP特效，可以用有趣的方式改变声音来模拟现实生活或夸大声音。

**开始**

介绍

FMOD Studio和低级API的设计非常直观和灵活。本节将介绍如何使用该引擎，以及如何有效地使用它所涉及的关键问题。

FMOD提供了一个C++ API和一个C API。它们是功能相同的，事实上，C++和C函数可以互换，C++和C类能够被反复地铸造。下面的示例只显示C++版本。

**初始化**

### **FMOD Studio API Initialization**

使用studioapi时，可以创建FMOD Studio System，然后调用Studio:：System:：initialize。该功能还将初始化内置的低级FMOD系统。下面是一个简单的例子：

[FMOD\_RESULT](mk:@MSITStore:D:\\PC-Loudspeaker\\fmod开发工具包\\FMOD%20Studio%20API%20Windows\\doc\\FMOD%20Studio%20Programmers%20API%20for%20Windows.chm::/content/generated/overview/../FMOD_RESULT.html) result;

FMOD::Studio::System\* system = NULL;

result = FMOD::Studio::System::create(&system); // Create the Studio System object.

if (result != [FMOD\_OK](mk:@MSITStore:D:\\PC-Loudspeaker\\fmod开发工具包\\FMOD%20Studio%20API%20Windows\\doc\\FMOD%20Studio%20Programmers%20API%20for%20Windows.chm::/content/generated/overview/../FMOD_RESULT.html))

{

printf("FMOD error! (%d) %s\n", result, FMOD\_ErrorString(result));

exit(-1);

}

// Initialize FMOD Studio, which will also initialize FMOD Low Level

result = system->initialize(512, [FMOD\_STUDIO\_INIT\_NORMAL](mk:@MSITStore:D:\\PC-Loudspeaker\\fmod开发工具包\\FMOD%20Studio%20API%20Windows\\doc\\FMOD%20Studio%20Programmers%20API%20for%20Windows.chm::/content/generated/overview/../FMOD_STUDIO_INITFLAGS.html), [FMOD\_INIT\_NORMAL](mk:@MSITStore:D:\\PC-Loudspeaker\\fmod开发工具包\\FMOD%20Studio%20API%20Windows\\doc\\FMOD%20Studio%20Programmers%20API%20for%20Windows.chm::/content/generated/overview/../FMOD_INITFLAGS.html), 0);

if (result != [FMOD\_OK](mk:@MSITStore:D:\\PC-Loudspeaker\\fmod开发工具包\\FMOD%20Studio%20API%20Windows\\doc\\FMOD%20Studio%20Programmers%20API%20for%20Windows.chm::/content/generated/overview/../FMOD_RESULT.html))

{

printf("FMOD error! (%d) %s\n", result, FMOD\_ErrorString(result));

exit(-1);

}

### **FMOD Low Level API Initialization (Do not use this if using FMOD Studio API Initialization)**

FMOD low level API可以完全不依赖FMOD Studio API使用。使用低级API可以访问基本功能，如加载和播放声音、创建DSP效果、设置FMOD通道组以及设置采样准确的衰减点和开始/停止时间。但是，当只使用低级API时，将无法加载Studio banks或加载并播放声音艺术家在Studio工具中设置的Studio事件。直接初始化FMOD low level：

[FMOD\_RESULT](mk:@MSITStore:D:\\PC-Loudspeaker\\fmod开发工具包\\FMOD%20Studio%20API%20Windows\\doc\\FMOD%20Studio%20Programmers%20API%20for%20Windows.chm::/content/generated/overview/../FMOD_RESULT.html) result;

FMOD::System \*system = NULL;

result = FMOD::System\_Create(&system); // Create the main system object.

if (result != [FMOD\_OK](mk:@MSITStore:D:\\PC-Loudspeaker\\fmod开发工具包\\FMOD%20Studio%20API%20Windows\\doc\\FMOD%20Studio%20Programmers%20API%20for%20Windows.chm::/content/generated/overview/../FMOD_RESULT.html))

{

printf("FMOD error! (%d) %s\n", result, FMOD\_ErrorString(result));

exit(-1);

}

result = system->init(512, [FMOD\_INIT\_NORMAL](mk:@MSITStore:D:\\PC-Loudspeaker\\fmod开发工具包\\FMOD%20Studio%20API%20Windows\\doc\\FMOD%20Studio%20Programmers%20API%20for%20Windows.chm::/content/generated/overview/../FMOD_INITFLAGS.html), 0); // Initialize FMOD.

if (result != [FMOD\_OK](mk:@MSITStore:D:\\PC-Loudspeaker\\fmod开发工具包\\FMOD%20Studio%20API%20Windows\\doc\\FMOD%20Studio%20Programmers%20API%20for%20Windows.chm::/content/generated/overview/../FMOD_RESULT.html))

{

printf("FMOD error! (%d) %s\n", result, FMOD\_ErrorString(result));

exit(-1);

}

**高级初始化**

FMOD可以通过在初始化前调用System:：setAdvancedSettings或Studio:：System:：setAdvancedSettings来自定义高级设置。有关有效虚拟语音的典型设置的说明，请参阅虚拟语音系统。

### **播放声音(Low Level API only)**

开始的最简单的方法，或者说FMOD低级API的基本功能，是初始化FMOD系统，加载一个声音，然后播放它。所有函数都会立即执行，因此开发人员要么在主循环执行期间触发并忘记，要么轮询声音以完成。播放声音不会锁定应用程序。

1. 使用上面描述的系统对象句柄，使用System:：createSound加载声音。这将返回一个声音的句柄。这是你的声音控制。
2. 使用步骤1返回的声音手柄，使用System:：playSound播放声音。这将返回一个channel句柄。
3. 让它在后台播放，或者用Channel:：isPlaying监视它的状态，使用从步骤2返回的通道句柄。当声音结束时，当调用任何相关的基于通道的函数时，通道句柄也会立即失效，所以这是另一种知道声音已结束的方法。返回的错误代码将是FMOD\_ERR\_INVALID\_HANDLE.