Managing Audio

This article is just a sampling. Any experienced modders that know more about audio playback, please feel free to flesh out this page.

Recommended Prerequisites

The recommended format for audio playback in HPL3 is the FMOD sound bank format. This is a special format that allows a number of sound files to be bundled together into a single library package as well as technical settings (such as reverb) to be easily accessible by the game engine. To create files in this format, download the FMOD Ex Designer software.

Also, to understand exactly how audio is handled in HPL3, you should go and read through Sound, as it covers the technical side in much greater detail than can be accomplished here.

Preparing FMOD Sound Banks

Replacing the .snt file format used in HPL2, an FMOD soundbank file is the preferred method of storing audio files for your mod. It allows sound to be played from within the game world, letting the player hear it in relation to where they are and where they are facing. It also lets you fine tune your audio in a number of different ways as well as define a number of game-engine specific properties.

This guide will show you how to create a very basic sound bank for use in your mod. First, start up FMOD Designer and create a new project. You can name it whatever you like, but it is recommended to give it a name that lets you and others know what kinds of audio files will be inside it.

When you first create the project, it will show you the following screen. (There will also be an event group created by default called “untitled”. Delete this event group for now via “Right Click > Delete”.)

The first thing we need to do is create an event group. An event group is a way to organize the sound events within your sound bank, similarly to how files are stored on your computer. You can create as many groups as you want, as well as creating groups within a group, but for now, let's just create a single group. Name it whatever you want, within reason.

Following this, we want to create the sound event itself. An event in FMOD represents a sound or sounds being played, and they contain most of the properties that you would want to set to a sound. (When you play a sound from within the game, you will be using the name of this event.)

Our event is currently empty, so we want to add a sound file to the playlist. This controls what sounds will be played when the event is triggered.

Our events are good for now, so next we want to create a sound definition. A sound definition represents the sound file itself. Click on the “Sound Defs” tab, right-click the window on the left, and select “Add sound def(s) from sound files…”. In the dialog box, select the sound file you chose for your event. All the relevant properties will be set for you, so that's all you need to do here.

When you created the project, a soundbank was created for you, but it is likely named something obnoxiously obtuse like “YourProjectName_001”. For the sake of clarity, you might want to rename it to something more helpful. Click on the “Banks” tab, select the soundbank from the list (of one) on the left, and rename it in the properties window on the right in the “Name” box. (This step is optional but recommended.)

Your soundbank is now complete (don't forget to save), but you still need to generate the files that HPL3 is actually going to use. To do this, go to the Project menu and select “Build…”. In the dialog, check the box next to your soundbank, then press “Build”.

You are now ready to use your newly-created soundbank within your mod. After all is said and done, the files in your sounds folder should look like this.

Most of these files are development or temporary files generated for you by FMOD Designer, but when you release your mod you only need the files ending in “.fdp” (the project file), “.fev” (the event-definition file), and “.fsb” (the soundbank file).

Playing FMOD Soundbanks

There are a number of functions provided by HPL3 to create and manage sounds played within the game. The game-related information relevant to the sound is either stored within the soundbank files or generated at run-time using Soundscape areas, so the functions themselves are relatively simple to use.

These functions are as follows:

```
//////////////////////////////////////////////////////////////////////////
////
// Remember to add "#include <helpers/helper_audio.hps>" to the top of your
// script file before using these functions.
```

```
// Basic playback functions
void Sound_CreateAtEntity(const tString &in asSoundName, const tString &in
asSoundFile, const tString &in asEntity, float afFadeTime = 0.0f, bool
abSaveSound = false, float afTargetVolume = 1.0f)
void Sound_Play(const tString &in asSoundName, float afFadeTime, bool
abResetVolMul = false)
void Sound_Stop(const tString &in asSoundName, float afFadeTime)
void Sound_Fade(const tString &in asSoundName, float afVolumeDest, float
afFadeTime)
bool Sound_Exists(const tString &in asSoundName)

// Advanced playback functions
Sound_CreateAtEntity_UsePrefix(const tString &in asSoundName, const tString
&in asSoundFile, const tString &in asEntity, float afFadeTime, bool
abSaveSound, float afTargetVolume = 1.0f)
```

These functions also support .snt files, but the .snt format has been deprecated by Frictional Games for use in HPL3. As such, not all features of .snt files are guaranteed to work, nor are they guaranteed to work everywhere. For this reason, it is recommended that you instead save your audio files in an FMOD soundbank.

Sound_CreateAtEntity

This function creates a sound dynamically, positioned at the specified entity, and plays the sound immediately. Sounds created in this way will be given a name, which can be used in other functions to affect a sound without creating a new one. (Useful for loops or sound effects.)

Using this function to create a sound from an FMOD soundbank requires that the sound file name is given in the following format: “SoundbankFileName/Event/Group/Names/EventName”.

- **asSoundname**: The name of the sound to be created.
- **asSoundFile**: The name of the sound file.
- **asEntity**: The name of the entity to create the sound at. Accepts the name “player”. (If the entity cannot be found, this function will do nothing.)
- **afFadeTime**: The time in seconds for the sound to fade in.
- **abSaveSound**: Whether or not the sound object should be saved when the game is saved. (Useful for sounds that start at the beginning of a level.)
- **afTargetVolume**: The volume at which the should be played. (0.0 - 1.0)

```
// Example Usage
Sound_CreateAtEntity("speaker_beep", "level_sounds/speaker/beep",
"intro_speaker");
```

Sound_Play

This function makes an existing sound object begin playing. The sound object can be placed in the level editor or have been created by a previous call to Sound_CreateAtEntity.

- **asSoundName**: The name of the sound to be played. (Supports wildcard characters (*) for playing many sounds at once.)
- **afFadeTime**: The time in seconds for the sound to fade in.
- **abResetVolMul**: Whether the volume for the sound should be reset to its default value.

```
// Example Usage
Sound_Play("explosion_bomb_*", 0.1);
```

Sound_Stop

This function stops the playing of a currently playing sound.

- **asSoundName**: The name of the sound to be stopped. (Supports wildcard characters (*) for stopping many sounds at once.)
- **afFadeTime**: The time in seconds for the sound to fade out.

```
// Example Usage
Sound_Stop("interrupting_cow", 0);

=== Sound_Fade ===
This function gradually changes the volume of a sound from its current
volume to a new one.

  * **asSoundName**: The name of the sound. (Supports wildcard characters
(*) for affecting many sounds at once.)
  * **afVolumeDest**: The target volume to fade to. (0.0 - 1.0)
  * **afFadeTime**: The time in seconds to fade from the old volume to the
new one.
<code>// Example Usage
Sound_Fade("ominous_noises_*", 0.25, 3.0);
```

Sound_Exists

This function checks whether a sound object exists.

- **asSoundName**: The name of the sound. (Supports wildcard characters (*) for checking many sounds at once.)
- **returns**: True if any sound object(s) that matches the given name exists, otherwise false.

```
// Example Usage
if (Sound_Exists("horse_noise_*"))
{
    // Magic happens here...
}
```

Sound_CreateAtEntity_UsePrefix

This function works similarly to Sound_CreateAtEntity with one major difference. If the sound object is created within a soundscape, this function will look for any specialized sound events in the soundbank for use within that soundscape. For example, if a sound with the event path “groupname/footstep” is created within the “smallroom” soundscape, this function will try to use the sound event named “groupname/smallroom/footstep”.

Using this function to create a sound from an FMOD soundbank requires that the sound file name is given in the following format: “SoundbankFileName/Event/Group/Names/EventName”.

- **asSoundname**: The name of the sound to be created.
- **asSoundFile**: The name of the sound file.
- **asEntity**: The name of the entity to create the sound at. Accepts the name “player”. (If the entity cannot be found, this function will do nothing.)
- **afFadeTime**: The time in seconds for the sound to fade in.
- **abSaveSound**: Whether or not the sound object should be saved when the game is saved. (Useful for sounds that start at the beginning of a level.)
- **afTargetVolume**: The volume at which the should be played. (0.0 - 1.0)

```
// Example Usage
Sound_CreateAtEntity_UsePrefix("footstep", "level_sounds/player/footstep",
"player");
```

Playing Music

WIP

Preparing Voice Files (using Voice Handler and VoiceHandler.exe)

Voice files are a specially handled form of audio in HPL3. Their main purpose is to represent the sound played from character voices, either in audio logs or in an active dialogue (such as the conversation with Dr. Munshi). They are also used to represent character barks (e.g. random sounds from monsters).

Unlike FMOD soundbanks, voice file audio sources are stored in a raw audio format, usually in a designated lang folder. The voice files themselves are special XML files that detail dialogue trees, handle subtitles, and generally manage the playback of audio files in character interactions.

Like FMOD soundbanks, voice files need some setup before they can be used in-game. The way they are created is with the VoiceHandler utility program that comes bundled with SOMA.

Overview - Setting up Voices for Mods:

1. Set up a standalone mod if you haven't already done so

2. Record voice lines and save them as .ogg

3. Create a .voice file (see “Creating a .voice file, step-by-step” below)

4. Create a “lang” folder in your mod directory, with the following path pattern:

```
SOMA\mods\[ExampleMod]\lang\eng\voices\[YourMapName]\[YourDialogFile].ogg
```

5. Rename each .ogg file of dialog to match the “File Name” in each voice handler line entry

- In VoiceHandler.exe under the “Subjects” tab, click the circular down arrow in a dialog line
- Next to the “File Name” box, click “Generate”, then “Copy”
- Rename the file to be [GeneratedName].ogg

6. In your map's .hps script file, trigger the “Subject” from your map's .voice file by adding this line (replacing “SubjectName” with the Subject that dialog line is located in when you made a .voice file):

```
Voice_Play("SubjectName");
```

7. (optional) Get “Test Voice Over” button to work

- Copy VoiceHandler.exe, libvorbis.dll, and libogg.dll from the SOMA directory to your mod's directory, and launch VoiceHandler.exe from your mod directory. Then open your .voice file and you should be able to preview Subjects

Creating a .voice file, step-by-step:

To start, we will need to open that program and create a new project.

For the purposes of this tutorial, I will be making a voice file dialogue out of this free audio clip, featuring a short one-sided conversation between an NPC in a game and a ghost. In order to use it in our dialogue, however, the clip must be split into a number of distinct audio files, each representing a particular line of dialogue. I won't be covering this step in detail, but using a free utility like Audacity this process is relatively straightforward.

The playback time of each line is directly based on the length of the associated audio clip. If you crop your audio clips so that the speech ends right at the end of the file, it may end up sounding too quick and unnatural in the game. When creating your clips, play with the length of silence at the end of the file to find a good point at which the dialogue sounds more natural.

Before we can create our dialogue, we need to create the scenes that the dialogue will be a part of, as well as any characters who participate in it. For this tutorial, we only have one scene, and our conversation only has one vocal participant.

Click on the “Voice Settings” tab and create a new voice scene and character. The scene only needs a name, and that can be whatever you like. The character has two fields that we care about: “Name” and “Display Name”. The Name is the name of the character that the voice file will use when deciding who is talking at the moment while the Display Name is the name that will appear in the subtitles.

Heading back to the “Subjects” tab, it's time to create our dialogue. In voice file vernacular, every chunk of dialogue is called a “Subject”, and each is handled as its own branch of conversation. Click the “Add Subject” button at the bottom-left of the window.

There are a lot of text boxes that pop-up with a new subject, but most of these aren't necessary for what we are currently doing, so ignore those for now. If you want, clicking the “Hide Options” box in the top-right will hide most of these boxes. (Don't do this now, however, as some of these boxes we will need later.)

While we aren't using them in this tutorial, the “Callback” and “Entity” boxes are very useful to fill out for your mod. The “Entity” box holds the name of an entity in your level that, when the subject is played, will automatically be set as the source of the voice audio. The “Callback” box holds the name of a callback function that you would like to be triggered when the subject ends.

The subject's name by default is “Subject0”, but we want to rename that into something a bit more useful. We are also going to want to select the “Scene” drop-down menu and select the scene that we created earlier.

Now it's time to create the dialogue itself. In the drop-down menu called “Select Character”, choose the character we created for this dialogue. Then in the box next to it, type out the text that subtitles the first line of the dialogue.

When that's finished, press the enter key to add a second line. Repeat this for all the lines in the dialogue.

The enter key will add a new line of dialogue for the currently speaking character. This is useful for breaking up a line of dialogue from a character that is too long. To create a line of dialogue for a new character, press the “+” button at the bottom of the subject.

Next, we need to hook up the lines of dialogue with the actual audio files. To do this, click the arrow button next to each line, and under “File Name” type the name of the audio file to be played. Do this for every line of dialogue.

That's all we need. Now it's time to save the voice file.

The most common place to save a voice file is in the same directory as the map files the voices will be played on. When doing this, name the voice file the same name as the map (for example, if your map is named “scary_map.hpm”, name the voice file “scary_map.voice”). This ensures that the voice file and the map will be paired correctly when the game is launched.

In order for the subtitles to be displayed, they need to be added to the lang file for your map. Fortunately, this process is handled for you. When you run your map after creating the voice file, the lang file will automatically be updated with the correct lang categories and entries. (This process may take a moment, so don't be surprised if the subtitles do not appear the first time you play your dialogue.)

Playing Voice Files

(This section is currently a work in progress.)

The easiest way to play a voice file is to use the Voice_Play function.

```
bool Voice_Play(const tString&in asSubject, int alSpecificLineIdx=-1, const tString&in asCallback="", int alPrio=0)
```

Voice_Play

This function starts the playback of a voice subject. Only one subject may be playing at any given time.

- **asSubject**: The name of the subject to play.
- **alSpecificLineIndex**: The line of dialogue to play in the subject. (If set to -1, all lines of dialogue will play according to the voice file.)
- **asCallback**: The name of a callback function to execute when the subject playback ends.
- **alPrio**: The priority value of this subject. (If the value given is higher than the priority value of the currently playing subject, the current subject will get interrupted in favor of this one, otherwise, nothing will happen.)

```
 // Example Usage
Voice_Play("OldMan_01", -1, "OldMan_SubjectComplete");
```

Playing Raw Audio Files

The simplest way to play an audio file is to do so directly. This has the benefit of having the most straight-forward implementation (just drop your audio files into your mod folder and you're done), but

it comes with the drawback of not being able to position the sound in the world. This could be the preferred method for audio playback during mod testing or in times when player position relative to the sound is unnecessary, such as when playing sound from a terminal or when the audio source is right in front of the player.

The relevant functions to achieve this are the following:

```
////////////////////////////////////////////////////////////////////////
////
// Remember to add "#include <helpers/helper_audio.hps>" to the top of your
// script file before using these functions.

// Basic playback functions
void Sound_PlayGui(const tString &in asSoundFile, float afVolume,
eSoundEntryType aEntryType = eSoundEntryType_Gui)
void Sound_FadeInGui(const tString &in asSoundFile, float afVolume, float
afFadeTime)
void Sound_StopGui(const tString &in asSoundFile, float afFadeTime, bool
abPlayEnd = true)
bool Sound_GuiIsPlaying(const tString &in asSoundFile)

// Advanced playback functions
void Sound_FadeGuiVolume(const tString &in asSoundFile, float afVolumeDest,
float afFadeTime)
void Sound_FadeGuiSpeed(const tString &in asSoundFile, float afSpeedDest,
float afFadeTime)
```

Sound_PlayGui

This is the base function for your audio-playing needs. It plays a sound from an FMOD sound bank, a .snt file, or a raw audio file. (.wav, .ogg, .mp3)

- **asSoundFile**: The path to your sound file.
- **afVolume**: The volume at which to play the sound. (0.0 - 1.0)
- **aEntryType**: The entry type of the sound. (It's usually best to ignore this parameter.)

```
// Example usage
Sound_PlayGui("sounds/button_beep.ogg", 1.0);
```

Sound_FadeInGui

Functionally similar to Sound_PlayGui, except it allows the option to specify a fade-in timer to your audio.

- **asSoundFile**: The path to your sound file.
- **afVolume**: The volume at which to play the sound. (0.0 - 1.0)
- **afFadeTime**: The time in seconds for the sound to fade in.

```
// Example Usage
```

```
Sound_PlayGui("sounds/building_tremor.ogg", 1.0, 3.5);
```

Sound_StopGui

This function allows you to manually stop the playback of an audio file. It also allows the option to specify a fade-out timer.

- **asSoundfile**: The path to your sound file. (Must be an audio file currently playing, or this function does nothing.)
- **afFadeTime**: The time in seconds for the sound to fade out.
- **abPlayEnd**: If you want end events to be triggered. (Only applicable if afFadeTime is 0. Not applicable to raw audio files.)

```
// Example Usage
Sound_StopGui("sounds/sudden_noises.ogg", 0.0);
```

Sound_GuiIsPlaying

This function lets you check to see if a sound file is currently playing or not.

- **asSoundfile**: The path to your sound file.
- **returns**: True if the specified sound file is currently playing, otherwise false.

```
// Example Usage
if (Sound_GuiIsPlaying("sound/quizzical_music.ogg))
{
    // Some magic happens here...
}
```

Sound_FadeGuiVolume

This function allows you to change the volume of a currently playing sound file.

- **asSoundfile**: The path to your sound file.
- **afVolumeDest**: The target volume to fade to. (0.0 - 1.0)
- **afFadeTime**: The time in seconds to fade from the old volume to the new volume.

```
// Example Usage
Sound_FadeGuiVolume("sound/be_quiet.ogg", 0.25, 0.75);
```

Sound_FadeGuiSpeed

This function lets you change the playback speed of a currently playing sound file. This operation does affect the pitch of the sound.

- **asSoundfile**: The path to your sound file.
- **afSpeedDest**: The target speed to fade to. (1.0 = Normal speed)
- **afFadeTime**: The time in seconds to fade from the old speed to the new speed.

```
// Example Usage
Sound_FadeGuiSpeed("sound/helium_talking.ogg", 2.0, 0.5);
```