XAudio2 BasicStream Sample

Collapse All

This sample shows you how to play streaming audio using asynchronous file I/O and the XAudio2 API.

Sample Location

Windows

The full working source code for this sample is in:

```
<Installed SDK Location>\Samples\C++\XAudio2\XAudio2BasicStream
```

Xbox 360

The full working source code for this sample is in:

```
<Installed XDK Location>\Source\Samples\Audio\XAudio2BasicStream
```

Sample Description

Playing streaming audio with XAudio2 is very easy. You do not need to create extra threads or maintain tight control of timing. Most of the hard work is already done for you by the file system and XAudio2. The file system handles loading data in the background, while XAudio2 manages the audio hardware, and makes sure that it receives a steady stream of data. The XAudio2 system even allows you to queue up data buffers, which eliminates the worry about getting too far ahead of the audio. Your only job is to direct this process.

First, you must decide on two important values:

1. The amount of data to request from the file system at a given time.
2. The number of data buffers to queue up for playback.

The exact values you use for these variables will depend on two factors: the amount of memory you want to use for streaming, and the load that the rest of your application places on the CPU and storage system. For instance, if other parts of your application make heavy use of the hard drive, then less hard drive capacity will be available for streaming audio. In this case, you will need to increase your buffer size or the number of buffers used.

Next, create the XAudio2 engine and a playback voice. The XAudio2BasicSound sample demonstrates how to do this.

After everything is set up, the streaming process is relatively straightforward:

1. Call **ReadFileEx** to start an asynchronous read.
2. When the read finishes, check the number of buffers in the playback queue.
3. As soon as the number of buffers in the playback queue falls below your predetermined maximum value, submit the data you received from the last asynchronous read.
4. Call **ReadFileEx** again to start another read.

There are two basic conditions that must be monitored during stream playback: the status of the file reads, and the number of buffers in the playback queue. Of course, it would be inefficient to poll these conditions continuously. There are many different ways to notify you when these conditions have changed, so that you can move to the next stage in the streaming process. This sample shows one of the simpler ways to track these conditions.

Note that in order to use asynchronous buffered file I/O, the source data must be aligned to the size of the disk sector. This is the fastest way to read data on both platforms. General .WAV files do not meet this requirement, so this sample uses XACTBLD to prepare the sample .WAV files into a wave bank. By using the streaming wave bank property, XACTBLD aligns the audio data to meet the alignment requirement. The sample parses the wave bank file by using synchronous operations to determine the format and position of the audio data. It then reopens the file for an asynchronous unbuffered read that uses this extracted information. The XACT runtime is not used by this sample. Any custom audio tool can provide audio data to meet the alignment requirement.