EN

此内容没有您的语言版本，但有英语版本。

Programming Guide

This section lists the overview topics for the XAudio2 application programming interface (API).

In This Section

|  |  |
| --- | --- |
| **Term** | **Description** |
| [XAudio2 Introduction](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415813(v=vs.85)) | Introduces key XAudio2 concepts. |
| [Common Audio Concepts](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415692(v=vs.85)) | Provides an overview of common audio concepts with which an audio developer should be familiar. |
| [Getting Started](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415762(v=vs.85)) | Describes setting up XAudio2 in game. |
| [Voices](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415824(v=vs.85)) | Describes the types of XAudio2 voices, and how to use them. |
| [Callbacks](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415743(v=vs.85)) | Provides details about the callbacks available for use with XAudio2. |
| [Audio Graph](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415739(v=vs.85)) | Describes the XAudio2 audio graph. |
| [Audio Effects](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415754(v=vs.85)) | Describes using digital signal processing effects with XAudio2. |
| [Streaming Audio Data](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415819(v=vs.85)) | Describes how to stream a sound from disk using XAudio2. |
| [X3DAudio Overview](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415714(v=vs.85)) | Describes how X3DAudio is used in conjunction with XAudio2 to create the illusion of a sound coming from a point in 3D space. |
| [XAudio2 Operation Sets](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415807(v=vs.85)) | Describes XAudio2 operation sets, and outlines some specific usage scenarios. |
| [Debugging Facilities](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415750(v=vs.85)) | Describes the XAudio2 debugging facilities. |
| [ADPCM Overview](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415711(v=vs.85)) | Describes the Adaptive Differential Pulse Code Modulation (ADPCM) compression format. |

Related topics

[XAudio2 Programming Reference](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415899(v=vs.85))

XAudio2 Introduction

XAudio2 is a low-level audio API. It provides a signal processing and mixing foundation for games that is similar to its predecessors, DirectSound and XAudio.

XAudio2 is the long-awaited replacement for DirectSound. It addresses several outstanding issues and feature requests.

XAudio2 Features

The following is a list of XAudio2 features and new functionality that enable developers to improve performance in their games.

* DSP Effects and Per Voice Filtering

Digital Signal Processing (DSP) effects are the pixel shaders of audio. They handle everything from transforming a sound—turning a pig squeal into a low, scary monster sound—to placing sounds in the game environment using reverb and occlusion or obstruction filtering. XAudio2 provides a flexible and powerful DSP framework. It also provides a built-in filter on every voice, for efficient low/high/band-pass filtering effects.

See [XAudio2 Audio Effects](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415756(v=vs.85)) and [**IXAudio2Voice::SetFilterParameters**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voice.ixaudio2voice.setfilterparameters(v=vs.85)) for more information about DSP effects and per voice filtering.

* Submixing

Submixing combines several sounds into a single audio stream—for example, an engine sound made up of composite parts, all of which are playing simultaneously. Also, you can use submixing to process and combine similar parts of a game. For example, you could combine all game sound effects to allow a user volume setting to be applied while a separate setting controls music volume. Combined with DSP, submixing provides the type of data routing and processing necessary for today's games. XAudio2 allows for arbitrary levels of submixing, enabling the creation of complex sounds and game mixes.

See [XAudio2 Audio Graph](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415741(v=vs.85)) and [XAudio2 Voices](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415825(v=vs.85)) for more information about submixing.

* Compressed Audio Support

One of the major feature requests for DirectSound has been for compressed audio support. XAudio2 supports compressed formats—ADPCM—natively with run-time decompression.

* Enhanced Multichannel and Surround Sound Support

Multichannel, 3D, and surround sound support is expanded. 3D and surround sound are now much more flexible and transparent. XAudio2 removes the 6-channel limit on multichannel sounds, and supports multichannel audio on any multichannel-capable audio card. The card does not need to be hardware-accelerated.

* Multirate Processing

To help minimize CPU usage, XAudio2 provides the technology to create multiple, low-rate audio processing graphs. This can significantly reduce CPU usage by allowing a game to process audio at the rate of the source material if the rate is less than 48 kHz.

* Nonblocking API Model

With few exceptions, an XAudio2 method call will not block the audio processing engine. This means that a client can safely make a set of method calls at any time without blocking on long-running calls causing delays. The exceptions are the [**IXAudio2Voice::DestroyVoice**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voice.ixaudio2voice.destroyvoice(v=vs.85)) method (which may block the engine until the voice being destroyed is finished processing) and the methods that terminate the audio thread: [**IXAudio2::StopEngine**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2.ixaudio2.stopengine(v=vs.85)) and[**IXAudio2::Release**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2.ixaudio2.release(v=vs.85)). Note that while XAudio2 method calls will not block the audio processing engine, the XAudio2 methods contain critical sections and may themselves become blocked in some circumstances.

When to use XAudio2

XAudio2 is primarily intended for developing high performance audio engines for games. For game developers who want to add sound effects and background music to their modern games, XAudio2 offers an audio graph and mixing engine with low-latency and support for dynamic buffers, synchronous sample-accurate playback, and implicit source rate conversion. Compared to WASAPI, XAudio2 requires only a minimum amount of code even for complex audio solutions. Compared to the Media Foundation engine, XAudio2 is a low-level, low-latency C++ API that is designed for use in games.

For applications that simply need regular music playback, the Media Foundation engine may be a better match to the application's requirements.

Related topics

[Getting Started](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415762(v=vs.85))

[XAudio2 Programming Reference](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415899(v=vs.85))

Common Audio Concepts

This section provides an overview of common audio concepts with which an audio developer should be familiar.

In This Section

|  |  |
| --- | --- |
| **Term** | **Description** |
| [Coordinates of 3D Space](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee416345(v=vs.85)) | Describes the coordinates of 3D space. |
| [Perception of Sound Positions](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee418717(v=vs.85)) | Describes the perceptions of different sound positions. |
| [Sound Cones](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee418803(v=vs.85)) | Describes sound cones. |

Coordinates of 3D Space

The position, velocity, and orientation of sound sources and listeners in 3D space are represented by Cartesian coordinates, which are values on three axes: the x-axis, the y-axis, and the z-axis.

The axes are relative to a viewpoint established by the application. Values on the x-axis increase from left to right, on the y-axis from down to up, and on the z-axis from near to far.

The **X3DAUDIO\_VECTOR** structure contains values describing position, velocity, or orientation on the three axes.

Conventionally, vectors are expressed as three values enclosed in parentheses and separated by commas, in the order (x, y, z).

For position, the values are in user-defined world units.

For velocity, the vector describes the rate of movement along each axis in world units per second.

For orientation, the values are in arbitrary units, and they are relative to one another. For example, if the base view of the 3D world is facing north toward the horizon, and the orientation of the listener is (-1, 0, 1), then the listener is facing northwest. Because the values within a vector are not in absolute units, the vector equally could be expressed as (-5, 0, 5) or (-0.25, 0, 0.25).

3D vectors work much like 2D vectors, but with an additional axis in the up-down direction. You can see how vectors work in 2D space by drawing them on a sheet of graph paper. Let the values increase from the bottom to the top of the paper, and from left to right. A line drawn from (0, 0) to (1, 1) has the same orientation, or direction, as one drawn from (0, 0) to (5, 5). However, the second line indicates a greater distance, or velocity.

Related topics

Perception of Sound Positions

Factors that influence the perception of sound positions.

In the real world clues from the sounds themselves include the following:

* Overall loudness. As a sound source moves away from the listener, its perceived volume decreases at a fixed rate. This phenomenon is known as rolloff.
* Interaural intensity difference. A sound coming from the listener's right sounds louder in the right ear than in the left.
* Interaural time difference. A sound emitted by a source to the listener's right will arrive at the right ear slightly before it arrives at the left ear. The duration of this offset is approximately a millisecond.
* Muffling. The shape and orientation of the ears ensures that sounds coming from behind the listener are slightly muffled compared with sounds coming from in front. Also, if a sound is coming from the right, the sound reaching the left ear will be muffled by the mass of the listener's head as well as by the orientation of the left ear.
* Effect of the earlobes. The pinnae, or folds of the ear, cause subtle changes to the pitch and timing of sounds arriving from different directions. The mathematics behind this effect are known as the head-related transfer function (HRTF).
* Sound Cones
* A model that describes the loudness of oriented sound.
* A sound with no orientation has the same amplitude at a given distance in all directions. A sound with an orientation is loudest in the direction of orientation. The model that describes the loudness of the oriented sound is called a sound cone. Sound cones are made up of an inside (or inner) cone and an outside (or outer) cone. The outside cone angle must always be equal to or greater than the inside cone angle.
* At any angle within the inner cone, the volume of the sound is set to the inner cone volume. This takes into account the basic volume of the buffer, the distance from the listener, the listener's orientation if the listener has its own cone, and so on.
* At any angle outside the outer cone, the normal volume is attenuated by a factor set by the application. The outside cone volume level is expressed as a linear amplitude scaler: 1.0 f represents no attenuation applied to the original signal, 0.5 f denotes an attenuation of 6 dB, and 0.0 f results in silence. Amplification (volume > 1.0 f) is also allowed, and is not clamped. The valid volume range is actually 0.0 f to 2.0 f.
* Between the inner and outer cones is a zone of transition from the inside volume to the outside volume. The volume approaches the cone's outer volume as the angle increases.
* Cones can affect parameters other than volume. Low pass filter and reverb send level may also be affected, making the technique even more dramatic. For example, with a cone on the listener, one can specify all sounds behind the listener get a bit muffled, and have slightly higher reverb-to-direct ratio content. These provide more cues that the sound is behind the user. This enhances realism.
* The following illustration shows the concept of sound cones.
* Designing sound cones properly can add dramatic effects to your application. For example, you could position a sound source in the center of a room, setting its orientation toward an open door in a hallway. Then set the angle of the inside cone so that it extends to the width of the doorway, make the outside cone a bit wider, and finally set the outside cone volume to inaudible. A listener moving along the hallway will begin to hear the sound only when near the doorway. The sound will be loudest as the listener passes in front of the open door.

# Getting Started

This section provides a list of key topics about XAudio2.

## In This Section

|  |  |
| --- | --- |
| **Term** | **Description** |
| [XAudio2 Key Concepts](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415764(v=vs.85)) | Introduces key concepts for using XAudio2. |
| [XAudio2 Versions](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415802(v=vs.85)) | Describes the versions of the XAudio2 libraries available. |
| [How to: Initialize XAudio2](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415779(v=vs.85)) | Describes the steps required to initialize XAudio2. |
| [Resource Interchange File Format (RIFF)](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415713(v=vs.85)) | Describes the Resource Interchange File Format (RIFF) used in audio data files. |
| [How to: Load Audio Data Files in XAudio2](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415781(v=vs.85)) | Describes the steps required to populate WAVEFORMATEX and XAUDIO2\_BUFFER structures with data from an audio file. |
| [How to: Play a Sound with XAudio2](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415787(v=vs.85)) | Describes the minimum steps required to play a sound using XAudio2. |

# XAudio2 Key Concepts

This overview introduces some key concepts for using XAudio2.

* [XAudio2 Engine](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415764(v=vs.85)#xaudio2_engine)
* [Voices](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415764(v=vs.85)#voices)
* [Audio Graph](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415764(v=vs.85)#audio_graph)
* [Callbacks](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415764(v=vs.85)#callbacks)
* [Related Topics](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415764(v=vs.85)#related_topics)

## XAudio2 Engine

The [**IXAudio2**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2.ixaudio2(v=vs.85)) interface is the core of the XAudio2 engine. Creating an instance of the **IXAudio2** interface allows the client to enumerate the available audio devices, to configure global API properties, to create voices, and to monitor performance. The [**XAudio2Create**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xaudio2.xaudio2create(v=vs.85)) helper function performs instantiation and initialization tasks for XAudio2.

You can create instances of XAudio2 multiple times within a single process. Each XAudio2 object operates independently, and has its own audio processing thread. Only the debug settings are shared. This is important on Windows where several different components may be loaded in a single process. For example, Internet Explorer might use multiple XAudio2 components simultaneously. Although it is possible to create multiple XAudio2 engine objects within a single client application, you should not pass information between their respective graphs.

For an example of initializing the XAudio2 engine, see [How to: Initialize XAudio2](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415779(v=vs.85)).

## Voices

Voices are the objects XAudio2 use to process, to manipulate, and to play audio data. There are three types of voices in XAudio2.

* [**Source Voices**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2sourcevoice.ixaudio2sourcevoice(v=vs.85))

Source voices represent a stream of audio data. Source voices send their data to other types of voices.

* [**Submix Voices**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2submixvoice.ixaudio2submixvoice(v=vs.85))

Submix voices perform some manipulation of audio data they receive. One example of audio data manipulation might be sample rate conversion. After a submix voice processes data, it passes that data to another submix voice or to a master voice.

* [**Mastering Voices**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2masteringvoice.ixaudio2masteringvoice(v=vs.85))

Mastering voices receive data from source voices and submix voices, and sends that data to the audio hardware.

See [XAudio2 Voices](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415824(v=vs.85)) for an overview of XAudio2 voices.

## Audio Graph

An audio graph is a collection of XAudio2 voices. Audio starts at one side of an audio graph in source voices, optionally passes through one or more submix voices, and ends at a mastering voice. An audio graph will contain a source voice for each sound currently playing, zero or more submix voices, and one mastering voice. The simplest audio graph, and the minimum needed to make a noise in XAudio2, is a single source voice outputting directly to a mastering voice. See [How to: Play a Sound with XAudio2](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415787(v=vs.85)) for an example of the minimum steps need to play a sound with XAudio2.

See [XAudio2 Audio Graph](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415739(v=vs.85)) for an overview of XAudio2 audio graphs.

## Callbacks

Callbacks are the mechanism XAudio2 uses to signal client code that some event has occurred in a voice or in the engine object. Because audio playback is asynchronous in the XAudio2 engine, callbacks provide the only way to determine when a sound is finished playing.

See [XAudio2 Callbacks](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415743(v=vs.85)) for an overview of XAudio2 callbacks.

# XAudio2 Versions

XAudio2 is a cross-platform API that has shipped for use on Xbox 360 as well as versions of Windows, including Windows XP, Windows Vista, Windows 7, and Windows 8. On Xbox 360, XAudio2 ships as a static library that is compiled into the main game executable. On Windows, XAudio2 is provided as a Dynamic Link Library (DLL) installed into the system folders of the Operating System.

## Current version of XAudio2 on Windows 10

XAudio2 version 2.9 ships as part of Windows 10, XAUDIO2\_9.DLL, alongside XAudio2.8 to support older applications, and does not require redistribution.

XAudio2.9 has been updated with the following changes:

* New creation flags: XAUDIO2\_DEBUG\_ENGINE, XAUDIO2\_STOP\_ENGINE\_WHEN\_IDLE, XAUDIO2\_1024\_QUANTUM
* xWMA support is available in this version of XAudio2.
* The [**CreateHrtfApo**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/mt186596(v=vs.85)) function is supported in this version of XAudio2.
* [**XAUDIO2FX\_REVERB\_PARAMETERS**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xaudio2.xaudio2fx_reverb_parameters(v=vs.85)) now includes the value SideDelay for 7.1 systems.
* The [**ReverbConvertI3DL2ToNative**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xaudio2.reverbconverti3dl2tonative(v=vs.85)) function now includes the boolean sevenDotOneReverb parameter enabling 7.1 reverb.

## XAudio 2.8 (Windows 8.x)

XAudio2 version 2.8 ships today as a system component in Windows 8, XAUDIO2\_8.DLL. It is available “inbox” and does not require redistribution with an app. We recommend to use the Windows Software Development Kit (SDK) for Windows 8 to develop against XAudio2; the Windows SDK for Windows 8 contains the necessary header and import library for statically linking against XAUDIO2\_8.DLL.

XAudio2 2.8 has been updated with the following changes:

* This version supports Windows Store app development; the XAudio2 API can be used in C++/DirectX Windows Store apps.
* [**XAudio2Create**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xaudio2.xaudio2create(v=vs.85)) is a flat Win32 API call and no longer creates an XAudio2 CLSID. Support for instantiating XAudio2 by CoCreateInstance has been removed.
* The Initialize function is now implicitly called by the creation process and has been removed from the [**IXAudio2**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2.ixaudio2(v=vs.85)) interface.
* Device enumeration functionality has been removed from XAudio2; the GetDeviceDetails and GetDeviceCount functions have been removed from the [**IXAudio2**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2.ixaudio2(v=vs.85)) interface. Apps that want to render to other audio devices on the system must pass a device identifier string to [**CreateMasteringVoice**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/hh405048(v=vs.85))instead of a device index. The default audio render device can still be created without enumeration.
* [**IXAudio2MasteringVoice**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2masteringvoice.ixaudio2masteringvoice(v=vs.85)) has an added function [**IXAudio2MasteringVoice::GetChannelMask**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2masteringvoice.ixaudio2masteringvoice.getchannelmask(v=vs.85)) for that returns the channel mask for the destination output device.
* The [X3DAudio](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415714(v=vs.85)) and [XAPOFX](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415726(v=vs.85)) libraries are merged into XAudio2. App code still uses separate headers, X3DAUDIO.H and XPOFX.H, but now links to a single import library, XAUDIO2\_8.LIB.
* xWMA support is not available in this version of XAudio2; xWMA will not be supported as an audio buffer format when calling CreateSourceVoice. We now recommend the Media Foundation Source Reader object for decoding a wide variety of media formats into in-memory PCM buffers.
* [**CreateFX**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/hh405044(v=vs.85)) now takes four parameters rather than two. The newer parameters specify initial data as part of [XAPOFX](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415726(v=vs.85)) creation.

## XAudio 2.7 and earlier (Windows 7)

All previous versions of XAudio2 for use in apps have been provided as redistributable DLLs in the DirectX SDK. The first version of XAudio2, XAudio2 2.0, shipped in the March 2008 release of the DirectX SDK. The last version to ship in the DirectX SDK was XAudio2 2.7, available in the last release of the DirectX SDK in June 2010.

For access to historical versions of XAudio2, download the DirectX SDK from the Microsoft Download Center. The June 2010 release of the DirectX SDK is available at:

[http://www.microsoft.com/download/en/details.aspx?id=6812](https://www.microsoft.com/download/en/details.aspx?id=6812)

Previous versions of XAudio2 cannot be used to build Windows Store apps for Windows 8.

# How to: Initialize XAudio2

XAudio2 is initialized for audio playback by creating an instance of the XAudio2 engine, and creating a mastering voice.

Ee415779.wedge(en-us,VS.85).gif**To initialize XAudio2**

1. Use the [**XAudio2Create**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xaudio2.xaudio2create(v=vs.85)) function to create an instance of the XAudio2 engine.
2. IXAudio2\* pXAudio2 = NULL;
3. HRESULT hr;
4. if ( FAILED(hr = XAudio2Create( &pXAudio2, 0, XAUDIO2\_DEFAULT\_PROCESSOR ) ) )
5. return hr;
6. Use the [**CreateMasteringVoice**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/hh405048(v=vs.85)) method to create a mastering voice.

The mastering voices encapsulates an audio device. It is the ultimate destination for all audio that passes through an audio graph.

IXAudio2MasteringVoice\* pMasterVoice = NULL;

if ( FAILED(hr = pXAudio2->CreateMasteringVoice( &pMasterVoice ) ) )

return hr;

## Notes for Windows Store apps

We recommend that you make use of a [smart pointer](https://msdn.microsoft.com/zh-cn/library/windows/desktop/hh279674(v=vs.85)) to manage the lifetime of XAUDIO2 objects in an exception safe manner. For Windows Store apps, you can use the [**ComPtr**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/br244983(v=vs.85)) smart pointer template from the Windows Runtime C++ Template Library (WRL).

C++

Microsoft::WRL::ComPtr<IXAudio2> XAudio2;

HRESULT hr;

if ( FAILED(hr = XAudio2Create( &XAudio2, 0, XAUDIO2\_DEFAULT\_PROCESSOR ) ) )

throw Platform::Exception::CreateException(hr);

IXAudio2MasteringVoice\* pMasterVoice = NULL;

if ( FAILED(hr = pXAudio2->CreateMasteringVoice( &pMasterVoice ) ) )

return hr;

**Note**  Ensure that all smart pointers to XAUDIO2 objects are fully released before you release the [**IXAudio2**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2.ixaudio2(v=vs.85)) object.

# Resource Interchange File Format (RIFF)

This overview describes the Resource Interchange File Format (RIFF), which is used in .wav files. RIFF is the typical format from which audio data for XAudio2 will be loaded.

## RIFF

A RIFF file is composed of multiple discrete sections of data called chunks.

## FOURCC Identifiers

The type of data in a chunk is indicated by a four-character code (FOURCC) identifier. A FOURCC is a 32-bit unsigned integer created by concatenating four ASCII characters used to identify chunk types in a RIFF file. For example, the FOURCC "abcd" is represented on a little-endian system as 0x64636261. FOURCCs can contain space characters, so " abc" is a valid FOURCC. Audio files use FOURCC codes to identify audio format chunks, audio data chunks, and any other chunks specific to the audio format.

The following table shows the FOURCC identifiers that can be expected in the audio formats supported by XAudio2.

|  |  |  |
| --- | --- | --- |
| **Format** | **FOURCC identifiers** | **Additional information** |
| PCM | "RIFF", "fmt" , "data" |  |
| ADPCM | "RIFF", "fmt", "data", "smpl", "wsmpl" | See [ADPCM Overview](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415711(v=vs.85)) for a description of the ADPCM-specific FOURCC identifiers. |

The FOURCC identifiers "RIFF", "fmt", and "data" are common to all of the supported formats. The following table describes the FOURCC identifiers that are found in all of the supported formats.

|  |  |
| --- | --- |
| **FOURCC identifier** | **Description** |
| "RIFF" | Standard RIFF chunk containing a file type with the value of "WAVE" or "XWMA" in the first four bytes of its data section and the other chunks in the file in the remainder of its data section. |
| "fmt" | Contains the format header for the audio file. The data in this chunk corresponds to one of the following structures:**WAVEFORMATEX**, **WAVEFORMATEXTENSIBLE ADPCMWAVEFORMAT**. |
| "data" | Contains audio data for the audio file. In XAudio2, the contents of the data chunk will be read into a buffer and passed to a source voice as the **pAudioData** member of an [**XAUDIO2\_BUFFER**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xaudio2.xaudio2_buffer(v=vs.85)) structure. |

## Chunks

A RIFF file consists of a RIFF chunk containing zero or more other chunks.

* The RIFF chunk has the following form:

"RIFF", fileSize, fileType, data

Where "RIFF" is the literal FOURCC code "RIFF", fileSize is a 4-byte value giving the size of the data in the file, and fileType is a FOURCC that identifies the specific file type. The value of fileSize includes the size of fileType FOURCC plus the size of the data that follows, but does not include the size of the "RIFF" FOURCC or the size of fileSize. The data consists of chunks in any order.

* Other chunks have the following form:
* chunkID, chunkSize, data

Where chunkID is a FOURCC that identifies the data contained in the chunk, chunkSize is a 4-byte value giving the size of the data section of the chunk, and data is zero or more bytes of data. The data is always padded to the nearest WORD boundary. chunkSize gives the size of the valid data in the chunk. It does not include the padding, the size of chunkID, or the size of chunkSize.

# How to: Load Audio Data Files in XAudio2

**Note**  This content applies only to desktop apps and will require revision to function in a Windows Store app. Please refer to the documentation for[**CreateFile2**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/hh449422(v=vs.85)), [**CreateEventEx**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ms682400(v=vs.85)), [**WaitForSingleObjectEx**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ms687036(v=vs.85)), [**SetFilePointerEx**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/aa365542(v=vs.85)), and [**GetOverlappedResultEx**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/hh448542(v=vs.85)). See SoundFileReader.h/.cpp in the BasicSound Windows 8 sample from the [Windows SDK Samples Gallery](http://code.msdn.microsoft.com/).

This topic describes the steps to populate the structures required to play audio data in XAudio2. The following steps load the 'fmt ' and 'data' chunks of an audio file, and uses them to populate a **WAVEFORMATEXTENSIBLE** structure and an [**XAUDIO2\_BUFFER**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xaudio2.xaudio2_buffer(v=vs.85)) structure.

* [Preparing to parse the audio file.](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415781(v=vs.85)#preparing_to_parse_the_audio_file)
* [Populating XAudio2 structures with the contents of RIFF chunks.](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415781(v=vs.85)#populating_xaudio2_structures_with_the_contents_of_riff_chunks)

## Preparing to parse the audio file

Audio files supported by XAudio2 use the Resource Interchange File Format (RIFF). RIFF is described in the [Resource Interchange File Format (RIFF)](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415713(v=vs.85))overview. Audio data in a RIFF file is loaded by finding the RIFF chunk and then looping through the chunk to find individual chunks contained in the RIFF chunk. The following functions are examples of code to find chunks and load data contained in the chunks.

* To find a chunk in a RIFF file:
* #ifdef \_XBOX //Big-Endian
* #define fourccRIFF 'RIFF'
* #define fourccDATA 'data'
* #define fourccFMT 'fmt '
* #define fourccWAVE 'WAVE'
* #define fourccXWMA 'XWMA'
* #define fourccDPDS 'dpds'
* #endif
* #ifndef \_XBOX //Little-Endian
* #define fourccRIFF 'FFIR'
* #define fourccDATA 'atad'
* #define fourccFMT ' tmf'
* #define fourccWAVE 'EVAW'
* #define fourccXWMA 'AMWX'
* #define fourccDPDS 'sdpd'
* #endif
* HRESULT FindChunk(HANDLE hFile, DWORD fourcc, DWORD & dwChunkSize, DWORD & dwChunkDataPosition)
* {
* HRESULT hr = S\_OK;
* if( INVALID\_SET\_FILE\_POINTER == SetFilePointer( hFile, 0, NULL, FILE\_BEGIN ) )
* return HRESULT\_FROM\_WIN32( GetLastError() );
* DWORD dwChunkType;
* DWORD dwChunkDataSize;
* DWORD dwRIFFDataSize = 0;
* DWORD dwFileType;
* DWORD bytesRead = 0;
* DWORD dwOffset = 0;
* while (hr == S\_OK)
* {
* DWORD dwRead;
* if( 0 == ReadFile( hFile, &dwChunkType, sizeof(DWORD), &dwRead, NULL ) )
* hr = HRESULT\_FROM\_WIN32( GetLastError() );
* if( 0 == ReadFile( hFile, &dwChunkDataSize, sizeof(DWORD), &dwRead, NULL ) )
* hr = HRESULT\_FROM\_WIN32( GetLastError() );
* switch (dwChunkType)
* {
* case fourccRIFF:
* dwRIFFDataSize = dwChunkDataSize;
* dwChunkDataSize = 4;
* if( 0 == ReadFile( hFile, &dwFileType, sizeof(DWORD), &dwRead, NULL ) )
* hr = HRESULT\_FROM\_WIN32( GetLastError() );
* break;
* default:
* if( INVALID\_SET\_FILE\_POINTER == SetFilePointer( hFile, dwChunkDataSize, NULL, FILE\_CURRENT ) )
* return HRESULT\_FROM\_WIN32( GetLastError() );
* }
* dwOffset += sizeof(DWORD) \* 2;
* if (dwChunkType == fourcc)
* {
* dwChunkSize = dwChunkDataSize;
* dwChunkDataPosition = dwOffset;
* return S\_OK;
* }
* dwOffset += dwChunkDataSize;
* if (bytesRead >= dwRIFFDataSize) return S\_FALSE;
* }
* return S\_OK;
* }
* To read data in a chunk after it has been located.

Once a desired chunk is found, its data can be read by adjusting the file pointer to the beginning of the data section of the chunk. A function to read the data from a chunk once it is found might look like this.

HRESULT ReadChunkData(HANDLE hFile, void \* buffer, DWORD buffersize, DWORD bufferoffset)

{

HRESULT hr = S\_OK;

if( INVALID\_SET\_FILE\_POINTER == SetFilePointer( hFile, bufferoffset, NULL, FILE\_BEGIN ) )

return HRESULT\_FROM\_WIN32( GetLastError() );

DWORD dwRead;

if( 0 == ReadFile( hFile, buffer, buffersize, &dwRead, NULL ) )

hr = HRESULT\_FROM\_WIN32( GetLastError() );

return hr;

}

## Populating XAudio2 structures with the contents of RIFF chunks

In order for XAudio2 to play audio with a source voice, it needs a **WAVEFORMATEX** structure and an [**XAUDIO2\_BUFFER**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xaudio2.xaudio2_buffer(v=vs.85)) structure. The**WAVEFORMATEX** structure may be a larger structure such as **WAVEFORMATEXTENSIBLE** that contains a **WAVEFORMATEX** structure as its first member. See the **WAVEFORMATEX** reference page for more information.

In this example a **WAVEFORMATEXTENSIBLE** is being used to allow loading of PCM audio files with more than two channels.

The following steps illustrate using the functions described above to populate a **WAVEFORMATEXTENSIBLE** structure and an [**XAUDIO2\_BUFFER**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xaudio2.xaudio2_buffer(v=vs.85))structure. In this case, the audio file being loaded contains PCM data, and will only contain a 'RIFF', 'fmt ', and 'data' chunk. Other formats may contain additional chunk types as described in [Resource Interchange File Format (RIFF)](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415713(v=vs.85)).

1. Declare **WAVEFORMATEXTENSIBLE** and [**XAUDIO2\_BUFFER**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xaudio2.xaudio2_buffer(v=vs.85)) structures.
2. WAVEFORMATEXTENSIBLE wfx = {0};
3. XAUDIO2\_BUFFER buffer = {0};
4. Open the audio file with CreateFile.
5. #ifdef \_XBOX
6. char \* strFileName = "game:\\media\\MusicMono.wav";
7. #else
8. TCHAR \* strFileName = \_TEXT("media\\MusicMono.wav");
9. #endif
10. // Open the file
11. HANDLE hFile = CreateFile(
12. strFileName,
13. GENERIC\_READ,
14. FILE\_SHARE\_READ,
15. NULL,
16. OPEN\_EXISTING,
17. 0,
18. NULL );
19. if( INVALID\_HANDLE\_VALUE == hFile )
20. return HRESULT\_FROM\_WIN32( GetLastError() );
21. if( INVALID\_SET\_FILE\_POINTER == SetFilePointer( hFile, 0, NULL, FILE\_BEGIN ) )
22. return HRESULT\_FROM\_WIN32( GetLastError() );
23. Locate the 'RIFF' chunk in the audio file, and check the file type.
24. DWORD dwChunkSize;
25. DWORD dwChunkPosition;
26. //check the file type, should be fourccWAVE or 'XWMA'
27. FindChunk(hFile,fourccRIFF,dwChunkSize, dwChunkPosition );
28. DWORD filetype;
29. ReadChunkData(hFile,&filetype,sizeof(DWORD),dwChunkPosition);
30. if (filetype != fourccWAVE)
31. return S\_FALSE;
32. Locate the 'fmt ' chunk, and copy its contents into a **WAVEFORMATEXTENSIBLE** structure.
33. FindChunk(hFile,fourccFMT, dwChunkSize, dwChunkPosition );
34. ReadChunkData(hFile, &wfx, dwChunkSize, dwChunkPosition );
35. Locate the 'data' chunk, and read its contents into a buffer.
36. //fill out the audio data buffer with the contents of the fourccDATA chunk
37. FindChunk(hFile,fourccDATA,dwChunkSize, dwChunkPosition );
38. BYTE \* pDataBuffer = new BYTE[dwChunkSize];
39. ReadChunkData(hFile, pDataBuffer, dwChunkSize, dwChunkPosition);
40. Populate an [**XAUDIO2\_BUFFER**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xaudio2.xaudio2_buffer(v=vs.85)) structure.
41. buffer.AudioBytes = dwChunkSize; //buffer containing audio data
42. buffer.pAudioData = pDataBuffer; //size of the audio buffer in bytes
43. buffer.Flags = XAUDIO2\_END\_OF\_STREAM; // tell the source voice not to expect any data after this buffer

# How to: Play a Sound with XAudio2

This topic describes the minimum steps required to play previously-loaded audio data in XAudio2. After you initialize XAudio2 (see [How to: Initialize XAudio2](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415779(v=vs.85))) and load the audio data (see How to: [How to: Load Audio Data Files in XAudio2](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415781(v=vs.85))), you can play a sound by creating a source voice and passing audio data to it.

## To play a sound

1. Initialize the XAudio2 engine by following the steps described in [How to: Initialize XAudio2](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415779(v=vs.85)).
2. Populate a [**WAVEFORMATEX**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ff538799(v=vs.85)) and [**XAUDIO2\_BUFFER**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xaudio2.xaudio2_buffer(v=vs.85)) structure by following the steps described in [How to: Load Audio Data Files in XAudio2](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415781(v=vs.85)).

**Note**  Depending on the format of the audio data, you may need to use a larger data structure containing a [**WAVEFORMATEX**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ff538799(v=vs.85)) structure in place of a **WAVEFORMATEX**. See the **WAVEFORMATEX** reference page for more information.

1. Create a source voice by calling the [**IXAudio2::CreateSourceVoice**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2.ixaudio2.createsourcevoice(v=vs.85)) method on an instance of the XAudio2 engine. The format of the voice is specified by the values set in a [**WAVEFORMATEX**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ff538799(v=vs.85)) structure.
2. IXAudio2SourceVoice\* pSourceVoice;
3. if( FAILED(hr = pXAudio2->CreateSourceVoice( &pSourceVoice, (WAVEFORMATEX\*)&wfx ) ) ) return hr;
4. Submit an [**XAUDIO2\_BUFFER**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xaudio2.xaudio2_buffer(v=vs.85)) to the source voice using the function [**SubmitSourceBuffer**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2sourcevoice.ixaudio2sourcevoice.submitsourcebuffer(v=vs.85)).
5. if( FAILED(hr = pSourceVoice->SubmitSourceBuffer( &buffer ) ) )
6. return hr;

**Note**  The audio sample data to which buffer points is still 'owned' by the app and must remain allocated and accessible until the sound stops playing.

1. Use the [**Start**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2sourcevoice.ixaudio2sourcevoice.start(v=vs.85)) function to start the source voice. Since all XAudio2 voices send their output to the mastering voice by default, audio from the source voice automatically makes its way to the audio device selected at initialization. In a more complicated audio graph, the source voice would have to specify the voice to which its output should be sent.
2. if ( FAILED(hr = pSourceVoice->Start( 0 ) ) )
3. return hr;

## Notes for Windows Store apps

We recommend that you make use of a [smart pointer](https://msdn.microsoft.com/zh-cn/library/windows/desktop/hh279674(v=vs.85)) to manage the lifetime of XAUDIO2 objects in an exception safe manner. For Windows Store apps, you can use the [**ComPtr**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/br244983(v=vs.85)) smart pointer template from the Windows Runtime C++ Template Library (WRL).

Microsoft::WRL::ComPtr<IXAudio2SourceVoice> SourceVoice;

HRESULT hr;

if( FAILED(hr = pXAudio2->CreateSourceVoice( &SourceVoice, (WAVEFORMATEX\*)&wfx ) ) )

throw Platform::Exception::CreateException(hr);

if( FAILED(hr = SourceVoice->SubmitSourceBuffer( &buffer ) ) )

throw Platform::Exception::CreateException(hr);

if ( FAILED(hr = SourceVoice->Start( 0 ) ) )

throw Platform::Exception::CreateException(hr);

**Note**  Ensure that all smart pointers to XAUDIO2 objects are fully released before you release the [**IXAudio2**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2.ixaudio2(v=vs.85)) object.

# Voices

This section contains an overview of XAudio2 voices.

[XAudio2 Voices](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415825(v=vs.85))

Introduces XAudio2 voices.

[How to: Use Submix Voices](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415794(v=vs.85))

Describes how to use XAudio2 submix voices.

[XAudio2 Sample Rate Conversions](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415817(v=vs.85))

Explains the process of sample rate conversion in XAudio2.

[XAudio2 Default Channel Mapping](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415748(v=vs.85))

Describes the XAudio2 default channel mapping.

[XAudio2 Volume and Pitch Control](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415828(v=vs.85))

Describes the available methods of controlling volume and pitch in XAudio2.

# XAudio2 Voices

There are three types of XAudio2 voice objects: source, submix, and mastering voices. Source voices operate on audio data provided by the client. Source and submix voices send their output to one or more submix or mastering voices. Submix and mastering voices mix the audio from all voices feeding them, and operate on the result. Mastering voices write audio data to an audio device.

## Actions Performed by All Voices

All voices perform the following actions in order on the audio that travels though them.

1. Overall volume adjustment, affecting all audio channels. See [**IXAudio2Voice::SetVolume**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voice.ixaudio2voice.setvolume(v=vs.85)).
2. An optional client-specified chain of one or more DSP effects, such as the built-in reverb or a user effect defined by the [**IXAPO**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixapo.ixapo(v=vs.85)) interface. See[XAudio2 Audio Effects](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415756(v=vs.85)).
3. Per-channel output volume adjustment. See [**IXAudio2Voice::SetChannelVolumes**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voice.ixaudio2voice.setchannelvolumes(v=vs.85)).
4. Separate matrix mix to each of the destination voices or to the audio output device for mastering voices. This mix changes the number of channels in the audio, if necessary.

## Source Voices

Use source voices to submit audio data into the XAudio2 processing pipeline. They are the entry points into the [XAudio2 Audio Graph](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415741(v=vs.85)). You must send voice data to a mastering voice to be heard, either directly or through intermediate submix voices.

In addition to the actions performed by all voices, source voices perform the following actions.

* If necessary, a decoder runs first to convert encoded source data to Pulse Code Modulation (PCM).
* A variable-rate sample rate conversion (SRC) converts the voice's source audio data to the sample rate expected by its destination voices, if necessary, and also supports dynamic pitch changes.
* An optional state-variable filter can be used to color the sound in various ways. See [**IXAudio2Voice::SetFilterParameters**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voice.ixaudio2voice.setfilterparameters(v=vs.85)).
* An optional filter can be applied to the voice's outputs. See [**IXAudio2Voice::SetOutputFilterParameters**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voice.ixaudio2voice.setoutputfilterparameters(v=vs.85)).

## Submix Voices

A submix voice is used primarily for performance improvements and effects processing. You cannot submit data buffers directly to submix voices. It will not be audible unless you submit it to a mastering voice. You can use a submix voice to ensure that a particular set of voice data is converted to the same format and to have a particular effect chain processed on the collective result.

In addition to the actions performed by all voices, submix voices perform the following actions.

* A fixed-rate SRC runs on the voice's output, if necessary, to convert the audio to the sample rate expected by its destination voices.
* An optional state-variable filter can be used to color the sound in various ways. See [**IXAudio2Voice::SetFilterParameters**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voice.ixaudio2voice.setfilterparameters(v=vs.85)).
* An optional filter can be applied to the voice's outputs. See [**IXAudio2Voice::SetOutputFilterParameters**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voice.ixaudio2voice.setoutputfilterparameters(v=vs.85)).

## Mastering Voices

Use a mastering voice to represent the audio output device. You cannot submit data buffers directly to mastering voices, but data submitted to other types of voices must go to a mastering voice to be heard.

In addition to the actions performed by all voices, mastering voices perform the following actions.

* If you create the mastering voice with an explicit InputSampleRate value that is not supported by the audio device, a fixed-rate SRC is used to convert to the closest sample rate supported by the device.
* Clip the final output audio, if it is required by the output device.

How to: Use Submix Voices

This topic shows you how you can set groups of voices to send their output to the same submix voice. This enables a single change to a submix voice to affect a whole group of voices.

1. Create a [**submix voice**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2submixvoice.ixaudio2submixvoice(v=vs.85)) to which all of the game's sound effect voices will send.
2. IXAudio2SubmixVoice \* pSFXSubmixVoice;
3. pXAudio2->CreateSubmixVoice(&pSFXSubmixVoice,1,44100,0,0,0,0);
4. Create an [**XAUDIO2\_VOICE\_SENDS**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xaudio2.xaudio2_voice_sends(v=vs.85)) structure that contains a reference to the submix voice.
5. XAUDIO2\_SEND\_DESCRIPTOR SFXSend = {0, pSFXSubmixVoice};
6. XAUDIO2\_VOICE\_SENDS SFXSendList = {1, &SFXSend};
7. Pass the [**XAUDIO2\_VOICE\_SENDS**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xaudio2.xaudio2_voice_sends(v=vs.85)) structure to new source voices as they are created.
8. IXAudio2SourceVoice\* pSFXSourceVoice;
9. if( FAILED(hr = pXaudio2->CreateSourceVoice( &pSFXSourceVoice, (WAVEFORMATEX\*)&wfx,
10. 0, XAUDIO2\_DEFAULT\_FREQ\_RATIO, pCallback, pSFXSendList, NULL ) ) )
11. return hr;
12. Apply changes to all sound effect voices by adjusting the submix voice.

In this example, changing the volume of the submix voice with the [**SetVolume**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voice.ixaudio2voice.setvolume(v=vs.85)) function effectively changes the volume of all voices that output to it.

pSFXSubmixVoice->SetVolume(0.1);

XAudio2 Sample Rate Conversions

XAudio2 voices can perform automatic sample rate conversions if their input sample rate is different from the input sample rate of their output voices.

Sample rate conversions follow these rules:

* Voice input sample rate is fixed.

Voices can only handle the input sample rate specified when they were created. For [**mastering voices**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2masteringvoice.ixaudio2masteringvoice(v=vs.85)) and [**submix voices**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2submixvoice.ixaudio2submixvoice(v=vs.85)), the input sample rate is specified with the *InputSampleRate* argument to the [**IXAudio2::CreateMasteringVoice**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/hh405048(v=vs.85)) and [**IXAudio2::CreateSubmixVoice**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2.ixaudio2.createsubmixvoice(v=vs.85)) functions. For source voices, the input sample rate of the voice is specified by the pSourceFormat argument to the [**IXAudio2::CreateSourceVoice**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2.ixaudio2.createsourcevoice(v=vs.85)) function.

* All of a voice's output voices must have the same input sample rate.

Voices can convert from their input sample rate to any output sample rate, but all of the voice's output voices must have the same input sample rate. For example, a voice could output to any number of voices with an input sample rate of 22 kHz. However, if that same voice had several output voices, each of which had a different input sample rate, the audio graph would not be valid.

* Sample rate conversion processing only occurs when necessary.

Converting audio data to a different sample rate incurs more processing overhead, which it is preferable to avoid. If a voice's input sample rate matches the input sample rate of its output voices, this conversion is not done and processing time is shortened.

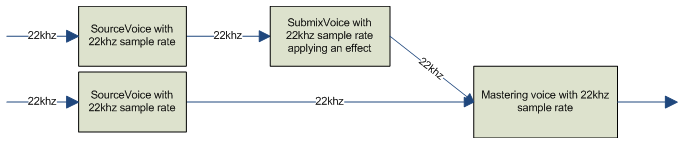
* Output sample rate can vary over the life of a voice.

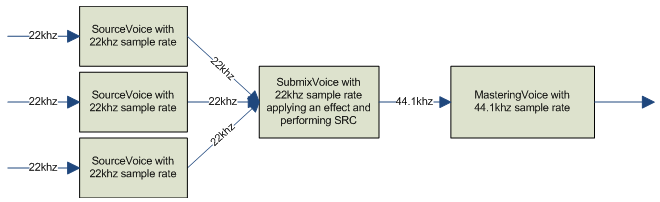
The output sample rate of a voice is not fixed. As long as all of its output voices have the same input sample rate, the audio graph will be valid. If a voice is changed to output to new voices with a different input sample rate, the voice will convert to the input sample rate of the new voices.

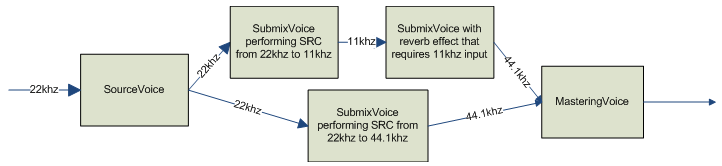
There are some scenarios in which it is necessary to add a submix voice to perform sample rate conversion between voices. If a voice needs to output to voices with various input sample rates, only one of the voices can be a direct output of the original voice. Because all of a voice's output voices must have the same input sample rate, the other voices receive output indirectly. There must be a submix voice with the correct input sample rate that comes between the original voice and the intended output voice.

For example, consider a source voice with an input sample rate of 22 kHz, which needs to output to a submix voice with an input sample rate of 11 kHz and a mastering voice with an input sample rate of 44.1 kHz. Because the two output voices have different input sample rates, you need to insert more submix voices between the original voice and its intended output voices. To maintain the fidelity of the source voice and avoid unnecessary expensive conversions to higher sample rates, you need to insert two submix voices with 22 khz sample input rates into the graph. One submix voice would output at 11 khz to the submix voice with the reverb effect, and the other submix voice would output to the mastering voice at 44.1 khz.

Examples of Sample Rate Conversion in Audio Graphs

All voices have the same sample input rate; no sample rate conversion is done in the audio graph.

All voices have the same sample input rate except the mastering voice; sample rate conversion is only performed on data going to the mastering voice.

Voices have different sample input rates and require more submix voices to perform sample rate conversions; sample rate conversion is performed in multiple places in the audio graph. 

XAudio2 Default Channel Mapping

An XAudio2 client has full control of the mapping from the channels of a voice to the channels of each of its destination voicesIt controls the mapping through the use of the [**IXAudio2Voice::SetOutputMatrix**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voice.ixaudio2voice.setoutputmatrix(v=vs.85)) method. In some circumstances, however, XAudio2 simplifies this task by setting up a default send matrix automatically. It does this by using the channel mask, if any, associated with a voice's audio channels. A channel mask is a combination of SPEAKER\_xxx bit masks as defined in X3DAudio.h and elsewhere. XAudio2 requires channel masks to be 0 or have the same number of bits set as the number of channels.

The following table shows the channel mask requirements and defaults for the formats supported by XAudio2.

|  |  |  |  |
| --- | --- | --- | --- |
| **Format** | **Channel Mask Requirement** | **Default Mask** | **Corresponding Structure Member** |
| PCM | File might contain a channel mask | Channel mask is 0, or absent | WAVEFORMATEXTENSIBLE.dwChannelMask or none (WAVEFORMATEX) |
| ADPCM | File does not contain a channel mask | Default Channel Mask is always used | None (ADPCMWAVEFORMAT) |

For submix and mastering voices, and for source voices without a channel mask or a channel mask of 0, XAudio2 assumes default speaker positions according to the following table.

|  |  |
| --- | --- |
| **Channels** | **Implicit Channel Positions** |
| 1 | Always maps to FrontLeft and FrontRight at full scale in both speakers (special case for mono sounds) |
| 2 | FrontLeft, FrontRight (basic stereo configuration) |
| 3 | FrontLeft, FrontRight, LowFrequency (2.1 configuration) |
| 4 | FrontLeft, FrontRight, BackLeft, BackRight (quadraphonic) |
| 5 | FrontLeft, FrontRight, FrontCenter, SideLeft, SideRight (5.0 configuration) |
| 6 | FrontLeft, FrontRight, FrontCenter, LowFrequency, SideLeft, SideRight (5.1 configuration) (see the following remarks) |
| 7 | FrontLeft, FrontRight, FrontCenter, LowFrequency, SideLeft, SideRight, BackCenter (6.1 configuration) |
| 8 | FrontLeft, FrontRight, FrontCenter, LowFrequency, BackLeft, BackRight, SideLeft, SideRight (7.1 configuration) |
| 9 or more | No implicit positions (one-to-one mapping) |

If a given voice pair in the audio graph has no speaker positions associated with either its source or target voice (one voice has more than eight channels), neither voice is playable until the source voice has a send matrix set explicitly using the [**IXAudio2Voice::SetOutputMatrix**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voice.ixaudio2voice.setoutputmatrix(v=vs.85)) method. Calling the [**IXAudio2SourceVoice::Start**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2sourcevoice.ixaudio2sourcevoice.start(v=vs.85)) method for either voice will fail until you do this.

If the source voice and target voice have different numbers of speaker positions and [**IXAudio2Voice::SetOutputMatrix**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voice.ixaudio2voice.setoutputmatrix(v=vs.85)) has not been called on the source voice, XAudio2 sends each source channel to the nearest target speaker (or speakers) available, proportionally to how close they are to the intended speaker. There are two special cases where the default behavior is different.

1. If the source audio is mono and is positioned at SPEAKER\_FRONT\_CENTER or has no defined position, it is always sent to SPEAKER\_FRONT\_LEFT and SPEAKER\_FRONT\_RIGHT if they exist in the output audio. If they do not exist, it falls back to the normal case.
2. If the source and destination are both 6-channel and are positioned at either of the standard 5.1 speaker setups (Left+Right+Center+Sub+BackL+BackR or Left+Right+Center+Sub+SideL+SideR), channels are mapped through one to one. In other words, SideLeft/Right and BackLeft/Right are treated equivalently. This is because there has been historical confusion around these setups. Therefore, the assumed intent is always to map one to one.

# XAudio2 Volume and Pitch Control

This topic describes XAudio2 volume and pitch control.

## Volume Control

Volume levels are expressed as floating-point amplitude multipliers between -XAUDIO2\_MAX\_VOLUME\_LEVEL and XAUDIO2\_MAX\_VOLUME\_LEVEL (-224 to 224), with a maximum gain of 144.5 dB. A volume of 1.0 means there is no attenuation or gain; 0 means silence; and negative levels can be used to invert the audio's phase. Two inline functions are provided in XAudio2.h to convert between volume units:[**XAudio2DecibelsToAmplitudeRatio**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xaudio2.xaudio2decibelstoamplituderatio(v=vs.85)) and [**XAudio2AmplitudeRatioToDecibels**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xaudio2.xaudio2amplituderatiotodecibels(v=vs.85)).

You can apply a volume level to the audio at several points as it flows through the XAudio2 graph:

* All voice types apply an overall volume level to their input, which they control using the [**IXAudio2Voice::SetVolume**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voice.ixaudio2voice.setvolume(v=vs.85)) method. In submix and mastering voices, the overall volume level is applied just before the voice's built-in filter and effect chain. In source voices, the overall volume level is applied after the voice's built-in filter and effect chain.
* Voices apply a per-channel volume level to their output, which they control using the [**IXAudio2Voice::SetChannelVolumes**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voice.ixaudio2voice.setchannelvolumes(v=vs.85)) method. The per-channel volume level is applied just after the voice's final sample rate conversion, and before it is sent to other voices.
* Every connection between one voice and another has a table of levels used to send audio from each source channel to each target channel, which is controlled using the [**IXAudio2Voice::SetOutputMatrix**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voice.ixaudio2voice.setoutputmatrix(v=vs.85)) method.

All overall volumes and channel volumes default to 1.0 initially. All send-level matrices default to appropriate values that preserve signal power and channel positioning as accurately as possible. See the [XAudio2 Default Channel Mapping](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415748(v=vs.85)) overview for details.

**Note**  XAudio2 automatically adjusts volume levels based on the user's speaker settings to maintain a consistent volume level across configurations. If the user's settings don't match their physical configuration the volume will either be too loud or too soft compared to a system with accurate settings. For example, a system configured for 5.1 surround sound speakers that only has two speakers connected will sound too soft. XAudio2 is unable to detect whether the user speaker settings correctly match their physical setup.

## Pitch Control

Pitches are expressed as input rate/output rate ratios between 1/1,024 and 1,024/1, inclusive. A ratio of 1/1,024 lowers pitch by 10 octaves, while a ratio of 1,024/1 raises it by 10 octaves. You can only use the [**IXAudio2SourceVoice::SetFrequencyRatio**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2sourcevoice.ixaudio2sourcevoice.setfrequencyratio(v=vs.85)) method to apply pitch adjustments to source voices, and only if they were not created with the XAUDIO2\_VOICE\_NOPITCH flag. The default frequency ratio is 1/1: that is, no pitch change. Two inline functions are provided in XAudio2.h to convert between frequency ratios and semitones: [**XAudio2FrequencyRatioToSemitones**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xaudio2.xaudio2frequencyratiotosemitones(v=vs.85)) and[**XAudio2SemitonesToFrequencyRatio**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xaudio2.xaudio2semitonestofrequencyratio(v=vs.85)).

# Callbacks

This section lists overview topics about XAudio2 callbacks.

[XAudio2 Callbacks](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415745(v=vs.85))

Introduces XAudio2 callbacks.

[How to: Use Source Voice Callbacks](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415769(v=vs.85))

Describes how to use XAudio2 source voice callbacks.

[How to: Use Engine Callbacks](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415774(v=vs.85))

Describes how to use XAudio2 engine callbacks.

XAudio2 Callbacks

XAudio2 can call functions provided by the client to notify it asynchronously of certain events taking place in the audio processing thread. These callbacks can be global or specific to a given source voice. To receive global engine callbacks, the client must provide an instance of a class implementing the [**IXAudio2EngineCallback**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2enginecallback.ixaudio2enginecallback(v=vs.85)) interface when initializing XAudio2. To receive source voice callbacks, the client must provide an instance of a class implementing the [**IXAudio2VoiceCallback**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voicecallback.ixaudio2voicecallback(v=vs.85)) interface when creating source voices. For more details, see **IXAudio2EngineCallback** and**IXAudio2VoiceCallback**.

You must implement callbacks carefully to avoid causing breaks in the audio. Whenever a callback is running, XAudio2 cannot generate any audio. Delays of more than a few milliseconds can cause an audio problem. Delays of this nature also generate debugger output. This indicates potential performance issues. At a minimum, callback functions must not do the following:

* Access the hard disk or other permanent storage
* Make expensive or blocking API calls
* Synchronize with other parts of client code
* Require significant CPU usage

If the client design requires a callback to trigger actions such as those listed previously, the callback should signal a different client thread to do the work. You can do this with a simple **SetEvent** mechanism or more sophisticated mechanisms like a nonblocking command queue that is consumed by another thread.

IXAudio2EngineCallback

The [**IXAudio2EngineCallback**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2enginecallback.ixaudio2enginecallback(v=vs.85)) class contains methods that notify the client when certain events happen in the XAudio2 engine. These methods should be implemented by the XAudio2 client. XAudio2 calls these methods by means of an interface pointer provided by the client using the[**IXAudio2::RegisterForCallbacks**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2.ixaudio2.registerforcallbacks(v=vs.85)) method. All these methods return **void**, rather than an **HRESULT**.

IXAudio2VoiceCallback

The [**IXAudio2VoiceCallback**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voicecallback.ixaudio2voicecallback(v=vs.85)) class contains methods that notify the client when certain events happen in a specific XAudio2 source voice. XAudio2 calls these methods by means of an interface pointer provided by the client in [**IXAudio2::CreateSourceVoice**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2.ixaudio2.createsourcevoice(v=vs.85)). As with [**IXAudio2EngineCallback**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2enginecallback.ixaudio2enginecallback(v=vs.85)), these methods should be implemented by the XAudio2 client, and return **void** rather than an **HRESULT**.

As mentioned previously, it is crucial that the client-provided implementations of these callbacks return as quickly as possible, preferably within a millisecond. The callbacks are executed in the audio processing thread, and all processing is interrupted until the callback returns. A delay in a callback can easily cause an audio problem.

How to: Use Source Voice Callbacks

When you create a source voice, you can pass a structure to it that defines callbacks for certain audio events. You can use these callbacks to perform actions or to signal other code.

1. Create a class that inherits from the [**IXAudio2VoiceCallback**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voicecallback.ixaudio2voicecallback(v=vs.85)) interface. All member functions of **IXAudio2VoiceCallback** are purely virtual, and must be defined. The only function of interest in this example is [**OnStreamEnd**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voicecallback.ixaudio2voicecallback.onstreamend(v=vs.85)). Therefore, the rest of the functions are stubs. The **OnStreamEnd** function triggers an event that indicates the sound is done playing.
2. class VoiceCallback : public IXAudio2VoiceCallback
3. {
4. public:
5. HANDLE hBufferEndEvent;
6. VoiceCallback(): hBufferEndEvent( CreateEvent( NULL, FALSE, FALSE, NULL ) ){}
7. ~VoiceCallback(){ CloseHandle( hBufferEndEvent ); }
8. //Called when the voice has just finished playing a contiguous audio stream.
9. void OnStreamEnd() { SetEvent( hBufferEndEvent ); }
10. //Unused methods are stubs
11. void OnVoiceProcessingPassEnd() { }
12. void OnVoiceProcessingPassStart(UINT32 SamplesRequired) { }
13. void OnBufferEnd(void \* pBufferContext) { }
14. void OnBufferStart(void \* pBufferContext) { }
15. void OnLoopEnd(void \* pBufferContext) { }
16. void OnVoiceError(void \* pBufferContext, HRESULT Error) { }
17. };
18. Create a [**source voice**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2sourcevoice.ixaudio2sourcevoice(v=vs.85)) with [**IXAudio2::CreateSourceVoice**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2.ixaudio2.createsourcevoice(v=vs.85)) using an instance of the callback class created previously as the pCallback parameter.
19. VoiceCallback voiceCallback;
20. if( FAILED(hr = pXaudio2->CreateSourceVoice( &pSourceVoice, (WAVEFORMATEX\*)&wfx,
21. 0, XAUDIO2\_DEFAULT\_FREQ\_RATIO, &voiceCallback, NULL, NULL ) ) ) return;
22. After starting the voice, use the [**WaitForSingleObjectEx**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ms687036(v=vs.85)) method to wait for the event to be triggered.

WaitForSingleObjectEx( voiceCallback.hBufferEndEvent, INFINITE, TRUE );

How to: Use Engine Callbacks

You can notify the XAudio2 client code of engine events by registering an instance of a class implementing the [**IXAudio2EngineCallback**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2enginecallback.ixaudio2enginecallback(v=vs.85)) interface with the XAudio2 engine. This allows the XAudio2 client code to keep track of when audio processing is occurring, and when to restart the engine in the event of a critical error.

To use an engine callback

The following steps register an object to handle engine events.

1. Create a class that inherits from the [**IXAudio2EngineCallback**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2enginecallback.ixaudio2enginecallback(v=vs.85)) interface.

All methods of [**IXAudio2EngineCallback**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2enginecallback.ixaudio2enginecallback(v=vs.85)) are purely virtual and must be defined. The method of interest in this example is[**IXAudio2EngineCallback::OnCriticalError**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2enginecallback.ixaudio2enginecallback.oncriticalerror(v=vs.85)), which sets a flag to signal the main game loop that a critical error has occurred. The remaining methods,[**IXAudio2EngineCallback::OnProcessingPassStart**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2enginecallback.ixaudio2enginecallback.onprocessingpassstart(v=vs.85)) and [**IXAudio2EngineCallback::OnProcessingPassEnd**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2enginecallback.ixaudio2enginecallback.onprocessingpassend(v=vs.85)), are stubs in this example.

class EngineCallback : public IXAudio2EngineCallback

{

void OnProcessingPassEnd () {}

void OnProcessingPassStart() {}

void OnCriticalError (HRESULT Error) {}

};

1. Use [**XAudio2Create**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xaudio2.xaudio2create(v=vs.85)) to create an instance of the XAudio2 engine.
2. if ( FAILED(hr = XAudio2Create( &pXAudio2, 0, XAUDIO2\_DEFAULT\_PROCESSOR ) ) )
3. return hr;
4. Use [**IXAudio2::RegisterForCallbacks**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2.ixaudio2.registerforcallbacks(v=vs.85)) to register the engine callback.
5. pXAudio2->RegisterForCallbacks( &engineCallback );
6. If you don't need the engine callback any more, call [**IXAudio2::UnregisterForCallbacks**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2.ixaudio2.unregisterforcallbacks(v=vs.85)).
7. pXAudio2->UnregisterForCallbacks( &engineCallback );

# Audio Graphs

This section lists the overview topics about the XAudio2 audio graph.

[XAudio2 Audio Graph](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415741(v=vs.85))

Introduces the XAudio2 audio graph.

[How to: Build a Basic Audio Processing Graph](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415767(v=vs.85))

Describes the minimum steps necessary to create an audio graph.

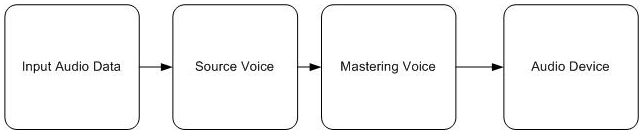
[How to: Dynamically Add or Remove Voices From an Audio Graph](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415772(v=vs.85))

Describes how to dynamically change an audio graph.

# XAudio2 Audio Graph

The set of all voices, with their contained effects and their interconnections, is referred to as the audio processing graph. The graph takes a set of audio streams from the client as input, processes them, and delivers the final result to an audio device. All audio processing takes place in a separate thread with a periodicity defined by the graph's quantum (currently 10 milliseconds on Microsoft Windows, and 5 1/3 milliseconds on Xbox 360). Every quantum milliseconds, the thread wakes up and disperses quantum milliseconds of audio data through the entire graph. For an example of building a basic audio graph, see How to: [Build a Basic Audio Processing Graph](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415767(v=vs.85)).

A simple audio graph:



The client can control the graph's state dynamically while it is running. Control actions might include adding and removing inputs and outputs, changing the internal effects and interconnections, setting parameters on the effects, enabling and disabling parts of the graph, and so on. For an example of dynamically changing an audio graph, see [How to: Dynamically Add or Remove Voices From an Audio Graph](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415772(v=vs.85)).

## Processing the Graph

Any method call that affects any object in the graph is considered to be effecting a graph state change. Graph state changes include the following:

* Creating and destroying voices
* Starting or stopping voices
* Changing the destinations of a voice
* Modifying effect chains
* Enabling or disabling effects
* Setting parameters on the effects or on the built-in SRCs, filters, volumes, and mixers

Any set of graph state changes can be combined and performed as an atomic transaction. These atomic operations are known as operation sets. They are discussed in the [XAudio2 Operation Sets](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415807(v=vs.85)) overview.

## Internal Data Representation

Audio data within the XAudio2 graph is always stored and processed in 32-bit floating-point PCM form. However, the channel count and sample rate can vary within the graph. The format in which a given voice processes audio is determined by the voice type and parameters used to create the voice.

|  |  |
| --- | --- |
| **Voice Type** | **Parameters** |
| [**IXAudio2SourceVoice**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2sourcevoice.ixaudio2sourcevoice(v=vs.85)) | The channel count and sample rate of the voices to which the source voice sends audio. |
| [**IXAudio2SubmixVoice**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2submixvoice.ixaudio2submixvoice(v=vs.85)) and[**IXAudio2MasteringVoice**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2masteringvoice.ixaudio2masteringvoice(v=vs.85)) | The InputChannels and InputSampleRate arguments used to create the submix/mastering voice. |

## Format Conversion

XAudio2 handles any sample rate or channel conversions that are required as audio travels from one voice to another, with the following limitations:

* All destination voices for a particular voice must be running at the same sample rate
* Effects in an effect chain can change the audio's channel count, but not its sample rate
* An effect chain's output channel count must match that of the voices to which it sends
* No dynamic graph change can be made which would break the rules above

On the input side, source voices can read data in any valid PCM format, or in any of the compressed formats supported by XAudio2. If the input data is compressed, it is decoded to floating-point PCM before any further processing is done.

On the output side, mastering voices can only produce PCM data. This data will always satisfy the same restrictions described above for input PCM data.

# How to: Build a Basic Audio Processing Graph

The minimum requirement for enabling XAudio2 to play audio data is an audio processing graph, which is constructed from a single mastering voice and a single source voice.

## To build a basic audio processing graph

1. Initialize the XAudio2 engine by following the steps described in [How to: Initialize XAudio2](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415779(v=vs.85)).
2. Populate a **WAVEFORMATEX** and [**XAUDIO2\_BUFFER**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xaudio2.xaudio2_buffer(v=vs.85)) structure by following the steps described in [How to: Load Audio Data Files in XAudio2](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415781(v=vs.85)).
3. Create a source voice using the [**CreateSourceVoice**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2.ixaudio2.createsourcevoice(v=vs.85)) function.

When you specify NULL for the pSendList argument of [**CreateSourceVoice**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2.ixaudio2.createsourcevoice(v=vs.85)), the source voice's output goes to the mastering voice created in step 1.

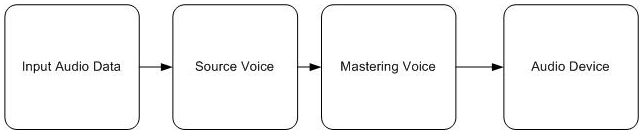
IXAudio2SourceVoice\* pSourceVoice;

if( FAILED(hr = pXAudio2->CreateSourceVoice( &pSourceVoice, (WAVEFORMATEX\*)&wfx,

0, XAUDIO2\_DEFAULT\_FREQ\_RATIO, NULL, NULL, NULL ) ) ) return hr;

After you finish this step, there is a simple audio graph consisting of the source voice, the mastering voice, and the audio device. The remaining steps in this how-to topic show you how to start audio data flowing through the graph.

A simple audio graph



1. Use the function [**SubmitSourceBuffer**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2sourcevoice.ixaudio2sourcevoice.submitsourcebuffer(v=vs.85)) to submit an [**XAUDIO2\_BUFFER**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xaudio2.xaudio2_buffer(v=vs.85)) to the source voice.
2. if( FAILED(hr = pSourceVoice->SubmitSourceBuffer( &buffer ) ) )
3. return hr;
4. Use the [**Start**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2sourcevoice.ixaudio2sourcevoice.start(v=vs.85)) function to start the source voice.
5. if ( FAILED(hr = pSourceVoice->Start( 0, XAUDIO2\_COMMIT\_NOW ) ) )

return hr;

# How to: Dynamically Add or Remove Voices From an Audio Graph

You can change audio graphs at any time to add or remove voices or entire subgraphs. This topic shows you how to add or remove submix voices from a graph that has been created following the steps in [How to: Build a Basic Audio Processing Graph](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415767(v=vs.85)). A single voice can send its output to several voices or to a long chain of voices. Removing or adding a single voice can have a large effect on an audio graph.

## To dynamically change an audio graph

Adding and removing voices from an audio graph is very similar to adding or removing nodes from a single-linked list or graph.

* To add a voice or subgraph to an audio graph

Set the output of a voice in the graph, the parent voice, to the voice to be added using the [**SetOutputVoices**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voice.ixaudio2voice.setoutputvoices(v=vs.85)) function. Set the output of the new voice to the original child of the parent voice.

XAUDIO2\_SEND\_DESCRIPTOR send = {0, pNewVoice};

XAUDIO2\_VOICE\_SENDS sendlist = {1, &send};

pParentVoice->SetOutputVoices(&sendlist);

send.pOutputVoice = pChildVoice;

pNewVoice->SetOutputVoices(&sendlist);

* To remove a voice or subgraph from an audio graph

Set the output voice of the parent of the voice being removed to the child of the voice being removed. If the voice being removed is at the end of the graph, the parent voice should be changed to point to the master voice.

XAUDIO2\_SEND\_DESCRIPTOR send = {0, pChildVoice};

XAUDIO2\_VOICE\_SENDS sendlist = {1, &send};

pParentVoice->SetOutputVoices(&sendlist);

Note that for clarity each parent only has one child in these examples. If a parent node has multiple children, its sendlist will contain an array of voices instead of a pointer to just one voice.

# Audio Effects

This section lists topics about XAudio2 audio effects.

[XAudio2 Audio Effects](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415756(v=vs.85))

Introduces XAudio2 audio effects.

[How to: Create an Effect Chain](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415789(v=vs.85))

Describes how to create an XAudio2 effect chain.

[XAPO Overview](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415735(v=vs.85))

Introduces concepts of cross-platform audio processing objects (XAPO).

[How to: Create an XAPO](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415730(v=vs.85))

Provides an example of creating a new XAPO.

[How to: Add Run-time Parameter Support to an XAPO](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415728(v=vs.85))

Provides an example of creating an XAPO that can have its behavior modified at run time.

[How to: Use an XAPO in XAudio2](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415733(v=vs.85))

Provides an example of how to use an XAPO in XAudio2.

[XAPOFX Overview](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415726(v=vs.85))

Introduces the cross-platform audio processing object effect library (XAPOFX).

[How to: Use XAPOFX in XAudio2](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415723(v=vs.85))

Provides an example of the steps needed to use XAPOFX in XAudio2.

# XAudio2 Audio Effects

An audio effect is an object that takes incoming audio data, and performs some operation on the data before passing it on. You can use an effect to perform a variety of tasks, including adding reverb to an audio stream and monitoring peak volume levels.

## Effect Chains

Any XAudio2 voice can host a chain of audio effects. You can use an array of [**XAUDIO2\_EFFECT\_DESCRIPTOR**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xaudio2.xaudio2_effect_descriptor(v=vs.85)) structures to specify effect chains. Each descriptor contains a pointer to an effect object provided by the client. These objects must implement the Audio Processing Object (APO) interfaces. See the [XAPO Overview](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415735(v=vs.85)) for more information about the APO model.

Effect chains can be modified by the client dynamically (while the XAudio2 engine is running), effects can be enabled or disabled individually, and effect parameters can be changed—all without any interruption of the audio. Whenever any aspect of the effect graph changes, XAudio2 optimizes the graph again to avoid unnecessary processing. See [**IXAudio2Voice::SetEffectChain**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voice.ixaudio2voice.seteffectchain(v=vs.85)), [**IXAudio2Voice::EnableEffect**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voice.ixaudio2voice.enableeffect(v=vs.85)), and[**IXAudio2Voice::SetEffectParameters**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voice.ixaudio2voice.seteffectparameters(v=vs.85)).

After an effect is attached to an XAudio2 voice, XAudio2 takes control of the effect, and the client should not make any further calls to it. The simplest way to ensure this is to release all pointers to the effect.

The effects in a given XAudio2 voice's effect chain must consume and produce floating-point audio at that voice's processing sample rate. The only aspect of the audio format they can change is the channel count (for example, a reverb effect can convert mono data to 5.1). The client can use the[**XAUDIO2\_EFFECT\_DESCRIPTOR**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xaudio2.xaudio2_effect_descriptor(v=vs.85)).**OutputChannels** field to specify the number of channels that each effect should produce. The effect chain fails if any of the effects cannot fulfill these requirements, or if an effect produces a number of channels that the next effect cannot handle. Any[**IXAudio2Voice::EnableEffect**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voice.ixaudio2voice.enableeffect(v=vs.85)) or [**IXAudio2Voice::DisableEffect**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voice.ixaudio2voice.disableeffect(v=vs.85)) calls that cause the effect chain to stop fulfilling these requirements will fail.

APO interfaces used in XAudio2 must be destructive. This means they always overwrite any data they find in their output buffers. Otherwise, the resulting audio might be incorrect because XAudio2 makes no guarantee that these buffers have been initialized previously with silence.

## XAudio2 Built-in Effects

The following table lists the set of built-in audio effects provided by XAudio2 and their creation methods.

|  |  |
| --- | --- |
| **Effect** | **Creation Method** |
| Reverb | [**XAudio2CreateReverb**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xaudio2.xaudio2createreverb(v=vs.85)) |
| Volume Meter | [**XAudio2CreateVolumeMeter**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xaudio2.xaudio2createvolumemeter(v=vs.85)) |

For an example of creating and using an instance of an audio effect, see [How to: Create an Effect Chain](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415789(v=vs.85)).

## Custom Effects in XAudio2

The [XAPO](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415735(v=vs.85)) API provides a framework for creating custom audio effects that you can use in XAudio2. For an example of creating a custom effect with XAPO, see [How to: Create an XAPO](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415730(v=vs.85)).

## XAPO Effect Library (XAPOFX)

[XAPOFX](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415726(v=vs.85)) provides an additional library of XAPOs and common mechanism for creating them. For an example of using XAPOFX with XAudio2, see [How to: Use XAPOFX in XAudio2](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415723(v=vs.85)).

# How to: Create an Effect Chain

This topic shows you how you can apply an effect chain to a voice to allow custom processing of the audio data for that voice. This topic describes how to use the reverb effect, which is one of the built-in XAudio2 effects.

## To create a basic effect chain that applies an effect to a voice

1. Create the effect.

In this example, the [**XAudio2CreateReverb**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xaudio2.xaudio2createreverb(v=vs.85)) function creates a reverb effect. See [XAudio2 Audio Effects](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415756(v=vs.85)) for a list of possible sources of effects for use with XAudio2.

IUnknown \* pXAPO;

hr = XAudio2CreateReverb(&pXAPO);

1. Populate an [**XAUDIO2\_EFFECT\_DESCRIPTOR**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xaudio2.xaudio2_effect_descriptor(v=vs.85)) structure with data.

If there are multiple effects in the chain, each effect will need a [**XAUDIO2\_EFFECT\_DESCRIPTOR**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xaudio2.xaudio2_effect_descriptor(v=vs.85)) structure.

XAUDIO2\_EFFECT\_DESCRIPTOR descriptor;

descriptor.InitialState = true;

descriptor.OutputChannels = 1;

descriptor.pEffect = pXAPO;

1. Populate an [**XAUDIO2\_EFFECT\_CHAIN**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xaudio2.xaudio2_effect_chain(v=vs.85)) structure with data. In this case, the chain only has one effect. If the chain has more than one effect, the EffectCount member will contain the count of effects, and the pEffectDescriptors member will point to an array of XAUDIO2\_EFFECT\_DESCRIPTOR structures.
2. XAUDIO2\_EFFECT\_CHAIN chain;
3. chain.EffectCount = 1;
4. chain.pEffectDescriptors = &descriptor;
5. Apply the effect chain to a voice with the [**SetEffectChain**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voice.ixaudio2voice.seteffectchain(v=vs.85)) function.

You can apply effect chains to master voices, source voices, and submix voices.

pVoice->SetEffectChain(&chain);

1. Release the effect with IUnknown::Release.

When you create an XAPO, it will have a reference count of 1. When the XAPO is passed to XAudio2 with [**SetEffectChain**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voice.ixaudio2voice.seteffectchain(v=vs.85)), XAudio2 increments the reference count on the XAPO. Releasing the client's reference to the XAPO allows XAudio2 to take ownership of the XAPO. If XAudio2 has the only reference to the XAPO, it will be disposed of when it is no longer being used by XAudio2. If the client code needs to maintain a reference to the XAPO—for example for later reuse—you should skip this step.

pXAPO->Release();

1. Populate the parameter structure, if any, associated with the effect. The reverb effect uses an [**XAUDIO2FX\_REVERB\_PARAMETERS**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xaudio2.xaudio2fx_reverb_parameters(v=vs.85)) structure.
2. XAUDIO2FX\_REVERB\_PARAMETERS reverbParameters;
3. reverbParameters.ReflectionsDelay = XAUDIO2FX\_REVERB\_DEFAULT\_REFLECTIONS\_DELAY;
4. reverbParameters.ReverbDelay = XAUDIO2FX\_REVERB\_DEFAULT\_REVERB\_DELAY;
5. reverbParameters.RearDelay = XAUDIO2FX\_REVERB\_DEFAULT\_REAR\_DELAY;
6. reverbParameters.PositionLeft = XAUDIO2FX\_REVERB\_DEFAULT\_POSITION;
7. reverbParameters.PositionRight = XAUDIO2FX\_REVERB\_DEFAULT\_POSITION;
8. reverbParameters.PositionMatrixLeft = XAUDIO2FX\_REVERB\_DEFAULT\_POSITION\_MATRIX;
9. reverbParameters.PositionMatrixRight = XAUDIO2FX\_REVERB\_DEFAULT\_POSITION\_MATRIX;
10. reverbParameters.EarlyDiffusion = XAUDIO2FX\_REVERB\_DEFAULT\_EARLY\_DIFFUSION;
11. reverbParameters.LateDiffusion = XAUDIO2FX\_REVERB\_DEFAULT\_LATE\_DIFFUSION;
12. reverbParameters.LowEQGain = XAUDIO2FX\_REVERB\_DEFAULT\_LOW\_EQ\_GAIN;
13. reverbParameters.LowEQCutoff = XAUDIO2FX\_REVERB\_DEFAULT\_LOW\_EQ\_CUTOFF;
14. reverbParameters.HighEQGain = XAUDIO2FX\_REVERB\_DEFAULT\_HIGH\_EQ\_GAIN;
15. reverbParameters.HighEQCutoff = XAUDIO2FX\_REVERB\_DEFAULT\_HIGH\_EQ\_CUTOFF;
16. reverbParameters.RoomFilterFreq = XAUDIO2FX\_REVERB\_DEFAULT\_ROOM\_FILTER\_FREQ;
17. reverbParameters.RoomFilterMain = XAUDIO2FX\_REVERB\_DEFAULT\_ROOM\_FILTER\_MAIN;
18. reverbParameters.RoomFilterHF = XAUDIO2FX\_REVERB\_DEFAULT\_ROOM\_FILTER\_HF;
19. reverbParameters.ReflectionsGain = XAUDIO2FX\_REVERB\_DEFAULT\_REFLECTIONS\_GAIN;
20. reverbParameters.ReverbGain = XAUDIO2FX\_REVERB\_DEFAULT\_REVERB\_GAIN;
21. reverbParameters.DecayTime = XAUDIO2FX\_REVERB\_DEFAULT\_DECAY\_TIME;
22. reverbParameters.Density = XAUDIO2FX\_REVERB\_DEFAULT\_DENSITY;
23. reverbParameters.RoomSize = XAUDIO2FX\_REVERB\_DEFAULT\_ROOM\_SIZE;
24. reverbParameters.WetDryMix = XAUDIO2FX\_REVERB\_DEFAULT\_WET\_DRY\_MIX;
25. Pass the effect parameter structure to the effect by calling the [**SetEffectParameters**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voice.ixaudio2voice.seteffectparameters(v=vs.85)) function on the voice to which the effect is attached.
26. hr = pVoice->SetEffectParameters( 0, &reverbParameters, sizeof( reverbParameters ) );
27. Disable or enable the effect, whenever appropriate.

You can use [**DisableEffect**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voice.ixaudio2voice.disableeffect(v=vs.85)) at any time to turn an effect off.

pVoice->DisableEffect(0);

You can turn on an effect again with [**EnableEffect**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voice.ixaudio2voice.enableeffect(v=vs.85)).

pVoice->EnableEffect(0);

The parameters for [**DisableEffect**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voice.ixaudio2voice.disableeffect(v=vs.85)) and [**EnableEffect**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voice.ixaudio2voice.enableeffect(v=vs.85)) specify which effect in the chain to enable or disable.

# XAPO Overview

The XAPO API allows the creation of cross-platform audio processing objects (XAPO) for use in XAudio2 on both Windows and Xbox 360. An XAPO is an object that takes incoming audio data, and performs some operation on the data before passing it on. You can use an XAPO to perform a variety of tasks, including adding reverb to an audio stream and monitoring peak volume levels.

## Creating New XAPOs

The XAPO API provides the [**IXAPO**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixapo.ixapo(v=vs.85)) interface and the [**CXAPOBase**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.cxapobase.cxapobase(v=vs.85)) class for building new XAPO types. The **IXAPO** interface contains all of the methods that need to be implemented to create a new XAPO. The **CXAPOBase** class provides a basic implementation of the **IXAPO** interface. **CXAPOBase**implements all of the **IXAPO** interface methods except the [**IXAPO::Process**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixapo.ixapo.process(v=vs.85)) method, which is unique to each XAPO.

For an example of creating a new XAPO, see [How to: Create an XAPO](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415730(v=vs.85)).

For an example of creating a XAPO that accepts run-time parameters, see [How to: Add Run-time Parameter Support to an XAPO](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415728(v=vs.85)).

## XAPOs and COM

XAPOs implement the **IUnknown** interface. The [**IXAPO**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixapo.ixapo(v=vs.85)) and [**IXAPOParameters**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixapoparameters.ixapoparameters(v=vs.85)) interfaces include the three **IUnknown** methods: **QueryInterface**,**AddRef**, and **Release**. [**CXAPOBase**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.cxapobase.cxapobase(v=vs.85)) provides implementations of all three of the IUnknown methods. A new instance of **CXAPOBase** will have a reference count of 1. It will be destroyed when its reference count becomes 0. Implementations of **IXAPO** and **IXAPOParameters** should follow the same pattern to allow for their proper management when used with XAudio2.

XAPO instances are passed to XAudio2 as **IUnknown** interfaces. XAudio2 uses **QueryInterface** to acquire an [**IXAPO**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixapo.ixapo(v=vs.85)) interface and to detect whether the XAPO implements the [**IXAPOParameters**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixapoparameters.ixapoparameters(v=vs.85)) interface. Implementations of **IXAPO** must accept requests for **\_\_uuidof(IXAPO)**. If **IXAPOParameters** is implemented, it must also accept requests for **\_\_uuidof(IXAPOParameters)**.

## Using an XAPO in XAudio2

XAPOs are used in XAudio2 by attaching them to voices. Each XAudio2 voice has an effect chain containing zero or more audio effects. Audio data sent to a voice is passed through each effect in the chain before it is sent to the voice's output targets. Data is passed from the voice to each effect using the pInputProcessParameters parameter of the [**IXAPO::Process**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixapo.ixapo.process(v=vs.85)) method. Then it is returned to the voice using the pOutputProcessParametersparameter. The voice takes the output of each effect, and feeds it into the next effect in the chain until no effects are left in the chain.

For more information about XAudio2 effect chains, see [XAudio2 Audio Effects](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415756(v=vs.85)).

For an example of using an XAPO in XAudio2, see [How to: Use an XAPO in XAudio2](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415733(v=vs.85)).

## Effect Libraries

The XAPO effect library contains several XAPOs, and a common method of instantiating them. See [XAPOFX Overview](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415726(v=vs.85)) for information about XAPOFX. Also, XAudio2 has built-in reverb and volume meter effects. See [XAudio2 Audio Effects](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415756(v=vs.85)) for more information about the built-in XAudio2 effects.

# How to: Create an XAPO

The XAPO API provides the [**IXAPO**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixapo.ixapo(v=vs.85)) interface and the [**CXAPOBase**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.cxapobase.cxapobase(v=vs.85)) class for building new XAPO types. The **IXAPO** interface contains all of the methods that need to be implemented to create a new XAPO. The **CXAPOBase** class provides a basic implementation of the **IXAPO** interface. **CXAPOBase**implements all of the **IXAPO** interface methods except the [**IXAPO::Process**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixapo.ixapo.process(v=vs.85)) method, which is unique to each XAPO.

## To create a new static XAPO

1. Derive a new XAPO class from the [**CXAPOBase**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.cxapobase.cxapobase(v=vs.85)) base class.

**Note**  XAPOs implement the **IUnknown** interface. The [**IXAPO**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixapo.ixapo(v=vs.85)) and [**IXAPOParameters**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixapoparameters.ixapoparameters(v=vs.85)) interfaces include the three **IUnknown** methods:**QueryInterface**, **AddRef**, and **Release**. [**CXAPOBase**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.cxapobase.cxapobase(v=vs.85)) provides implementations of all three of the **IUnknown** methods. A new instance of**CXAPOBase** will have a reference count of 1. It will be destroyed when its reference count becomes 0. To allow XAudio2 to destroy an instance of an XAPO when it is no longer needed, call **IUnknown::Release** on the XAPO after it is added to an XAudio2 effect chain. See [How to: Use an XAPO in XAudio2](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415733(v=vs.85)) for more information about using an XAPO with XAudio2.

1. Override the [**CXAPOBase**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.cxapobase.cxapobase(v=vs.85)) class implementation of the [**IXAPO::LockForProcess**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixapo.ixapo.lockforprocess(v=vs.85)) method.

Overriding [**LockForProcess**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixapo.ixapo.lockforprocess(v=vs.85)) allows information about the format of audio data to be stored for use in [**IXAPO::Process**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixapo.ixapo.process(v=vs.85)).

1. Implement the [**IXAPO::Process**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixapo.ixapo.process(v=vs.85)) method.

XAudio2 calls the [**IXAPO::Process**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixapo.ixapo.process(v=vs.85)) method whenever an XAPO needs to process audio data. **Process** contains the bulk of the code for an XAPO.

## Implementing the Process Method

The [**IXAPO::Process**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixapo.ixapo.process(v=vs.85)) method accepts stream buffers for input and output audio data. A typical XAPO will expect one input stream buffer and one output stream buffer. You should base the processing of data from the input stream buffer on the format specified in the [**LockForProcess**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixapo.ixapo.lockforprocess(v=vs.85)) function and any flags passed to the **Process** function with the input stream buffer. Copy the processed input stream buffer data to the output stream buffer. Set the output stream buffer's BufferFlags parameter as either [**XAPO\_BUFFER\_VALID**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xapo.xapo_buffer_flags(v=vs.85)) or **XAPO\_BUFFER\_SILENT**.

The following example demonstrates a [**LockForProcess**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixapo.ixapo.lockforprocess(v=vs.85)) and [**Process**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixapo.ixapo.process(v=vs.85)) implementation that simply copies data from an input buffer to an output buffer. However, there is no processing if the input buffer is marked with [**XAPO\_BUFFER\_SILENT**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xapo.xapo_buffer_flags(v=vs.85)).

STDMETHOD(LockForProcess) (UINT32 InputLockedParameterCount,

const XAPO\_LOCKFORPROCESS\_BUFFER\_PARAMETERS\* pInputLockedParameters,

UINT32 OutputLockedParameterCount,

const XAPO\_LOCKFORPROCESS\_BUFFER\_PARAMETERS\* pOutputLockedParameters)

{

assert(!IsLocked());

assert(InputLockedParameterCount == 1);

assert(OutputLockedParameterCount == 1);

assert(pInputLockedParameters != NULL);

assert(pOutputLockedParameters != NULL);

assert(pInputLockedParameters[0].pFormat != NULL);

assert(pOutputLockedParameters[0].pFormat != NULL);

m\_uChannels = pInputLockedParameters[0].pFormat->nChannels;

m\_uBytesPerSample = (pInputLockedParameters[0].pFormat->wBitsPerSample >> 3);

return CXAPOBase::LockForProcess(

InputLockedParameterCount,

pInputLockedParameters,

OutputLockedParameterCount,

pOutputLockedParameters);

}

STDMETHOD\_(void, Process)(UINT32 InputProcessParameterCount,

const XAPO\_PROCESS\_BUFFER\_PARAMETERS\* pInputProcessParameters,

UINT32 OutputProcessParameterCount,

XAPO\_PROCESS\_BUFFER\_PARAMETERS\* pOutputProcessParameters,

BOOL IsEnabled)

{

assert(IsLocked());

assert(InputProcessParameterCount == 1);

assert(OutputProcessParameterCount == 1);

assert(NULL != pInputProcessParameters);

assert(NULL != pOutputProcessParameters);

XAPO\_BUFFER\_FLAGS inFlags = pInputProcessParameters[0].BufferFlags;

XAPO\_BUFFER\_FLAGS outFlags = pOutputProcessParameters[0].BufferFlags;

// assert buffer flags are legitimate

assert(inFlags == XAPO\_BUFFER\_VALID || inFlags == XAPO\_BUFFER\_SILENT);

assert(outFlags == XAPO\_BUFFER\_VALID || outFlags == XAPO\_BUFFER\_SILENT);

// check input APO\_BUFFER\_FLAGS

switch (inFlags)

{

case XAPO\_BUFFER\_VALID:

{

void\* pvSrc = pInputProcessParameters[0].pBuffer;

assert(pvSrc != NULL);

void\* pvDst = pOutputProcessParameters[0].pBuffer;

assert(pvDst != NULL);

memcpy(pvDst,pvSrc,pInputProcessParameters[0].ValidFrameCount \* m\_uChannels \* m\_uBytesPerSample);

break;

}

case XAPO\_BUFFER\_SILENT:

{

// All that needs to be done for this case is setting the

// output buffer flag to XAPO\_BUFFER\_SILENT which is done below.

break;

}

}

// set destination valid frame count, and buffer flags

pOutputProcessParameters[0].ValidFrameCount = pInputProcessParameters[0].ValidFrameCount; // set destination frame count same as source

pOutputProcessParameters[0].BufferFlags = pInputProcessParameters[0].BufferFlags; // set destination buffer flags same as source

}

When writing a [**Process**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixapo.ixapo.process(v=vs.85)) method, it is important to note XAudio2 audio data is interleaved. This means data from each channel is adjacent for a particular sample number. For example, if there is a 4-channel wave playing into an XAudio2 source voice, the audio data is a sample of channel 0, a sample of channel 1, a sample of channel 2, a sample of channel 3, and then the next sample of channels 0, 1, 2, 3, and so on.

# How to: Add Run-time Parameter Support to an XAPO

You can add run-time parameter support to an XAPO by implementing the [**IXAPOParameters**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixapoparameters.ixapoparameters(v=vs.85)) interface. Run-time parameter support allows an XAPO to change its behavior based on the parameters passed to it at run time.

1. Follow the steps in [How to: Create an XAPO](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415730(v=vs.85)).
2. Change the XAPO to derive from [**CXAPOParametersBase**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.cxapoparameterbase.cxapoparametersbase(v=vs.85)) and [**CXAPOBase**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.cxapobase.cxapobase(v=vs.85)).
3. Add calls to the methods [**CXAPOParametersBase::BeginProcess**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.cxapoparameterbase.cxapoparametersbase.beginprocess(v=vs.85)) and [**CXAPOParametersBase::EndProcess**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.cxapoparameterbase.cxapoparametersbase.endprocess(v=vs.85)) to the implementation of[**IXAPO::Process**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixapo.ixapo.process(v=vs.85)).

**Note**  Adding these methods to [IXAPO::Process](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415767(v=vs.85)) allows [**CXAPOParametersBase**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.cxapoparameterbase.cxapoparametersbase(v=vs.85)) to keep its copies of the effect parameters in a thread-safe state. Call [**CXAPOParametersBase::BeginProcess**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.cxapoparameterbase.cxapoparametersbase.beginprocess(v=vs.85)) at the beginning of [**IXAPO::Process**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixapo.ixapo.process(v=vs.85)), and [**CXAPOParametersBase::EndProcess**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.cxapoparameterbase.cxapoparametersbase.endprocess(v=vs.85)) at the end of**IXAPO::Process**.

1. Add more code to the [**IXAPO::Process**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixapo.ixapo.process(v=vs.85)) implementation to change its behavior according to values stored by the [**SetParameters**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixapoparameters.ixapoparameters.setparameters(v=vs.85)) method.

**Note**  Adding code to the [**IXAPO::Process**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixapo.ixapo.process(v=vs.85)) method to use the parameters specified by [**SetParameters**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixapoparameters.ixapoparameters.setparameters(v=vs.85)) allows the XAPO's behavior to be changed throughout its life.

1. When you create an instance of the effect, allocate a buffer of three of the structures that will represent the effect's parameters, and pass it to the[**CXAPOParametersBase**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.cxapoparameterbase.cxapoparametersbase(v=vs.85)) constructor.

**Note**  The [**CXAPOParametersBase**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.cxapoparameterbase.cxapoparametersbase(v=vs.85)) instance internally uses this buffer to manage effect parameters passed to it when you call [**SetParameters**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixapoparameters.ixapoparameters.setparameters(v=vs.85)). You must initialize all the process parameter blocks in pParameterBlocks to the same default value before you call any of the [**IXAPO::Process**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixapo.ixapo.process(v=vs.85)),[**IXAPOParameters::GetParameters**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixapoparameters.ixapoparameters.getparameters(v=vs.85)), and **IXAPOParameters::SetParameters** methods. Usually this initialization is handled in [**IXAPO::Initialize**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixapo.ixapo.initialize(v=vs.85)) or in[**IXAPO::LockForProcess**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixapo.ixapo.lockforprocess(v=vs.85)).

# How to: Use an XAPO in XAudio2

This topic shows you how to use an effect created with the XAPO API in an XAudio2 effect chain.

1. Create the XAPO as described in [How to: Create an XAPO](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415730(v=vs.85)).

You can also implement run-time parameter functionality as described in [How to: Add Run-time Parameter Support to an XAPO](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415728(v=vs.85)).

1. Create an instance of the XAPO.
2. IUnknown \* pXAPO;
3. pXAPO = new SimpleXAPO();
4. Populate an [**XAUDIO2\_EFFECT\_DESCRIPTOR**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xaudio2.xaudio2_effect_descriptor(v=vs.85)) structure with data.
5. XAUDIO2\_EFFECT\_DESCRIPTOR descriptor;
6. descriptor.InitialState = true;
7. descriptor.OutputChannels = 1;
8. descriptor.pEffect = pXAPO;
9. Populate an [**XAUDIO2\_EFFECT\_CHAIN**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xaudio2.xaudio2_effect_chain(v=vs.85)) structure with data.
10. XAUDIO2\_EFFECT\_CHAIN chain;
11. chain.EffectCount = 1;
12. chain.pEffectDescriptors = &descriptor;
13. Apply the effect chain to an XAudio2 voice with the [**SetEffectChain**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voice.ixaudio2voice.seteffectchain(v=vs.85)) function.
14. pVoice->SetEffectChain(&chain);

**Note**  An effect chain can also be applied to a voice when the voice is created by passing the chain as a parameter to[**IXAudio2::CreateSourceVoice**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2.ixaudio2.createsourcevoice(v=vs.85)), [**IXAudio2::CreateSubmixVoice**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2.ixaudio2.createsubmixvoice(v=vs.85)), or [**IXAudio2::CreateMasteringVoice**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/hh405048(v=vs.85)).

1. Release the effect with IUnknown::Release.

When you create an XAPO, it will have a reference count of 1. When the XAPO is passed to XAudio2 with [**SetEffectChain**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voice.ixaudio2voice.seteffectchain(v=vs.85)), XAudio2 increments the reference count on the XAPO. Releasing the client's reference to the XAPO allows XAudio2 to take ownership of the XAPO. If XAudio2 has the only reference to the XAPO, it will be disposed of when it is no longer being used by XAudio2. If the client code needs to maintain a reference to the XAPO for later reuse, for example, you should skip this step.

pXAPO->Release();

1. Populate the parameter structure, if any, associated with the effect. In this case, the percentage of full strength at which the effect should be applied.
2. XAPO\_PARAMETERS XAPOParameters;
3. XAPOParameters.Level = 0.75;
4. Pass the effect parameter structure to the effect by calling the [**SetEffectParameters**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voice.ixaudio2voice.seteffectparameters(v=vs.85)) function on the voice to which the effect is attached.
5. hr = pVoice->SetEffectParameters( 0, &XAPOParameters, sizeof( XAPO\_PARAMETERS ) );

# XAPOFX Overview

XAPOFX is a collection of audio effects implementing the [XAPO](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415735(v=vs.85)) interfaces for use in XAudio2. XAPOFX contains several effects, and a common mechanism for creating effect instances.

## Included Effects

The following table describes the effects included in XAPOFX.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Effect** | **Description** | **Parameter Structure** | **Parameter Constants** | **Requirements** |
| FXECHO | An echo effect. | [**FXECHO\_PARAMETERS**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xapofx.fxecho_parameters(v=vs.85)) | [**FXECHO Constants**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee417668(v=vs.85)) | Only supports FLOAT32 audio formats. |
| FXEQ | A four band equalizer. | [**FXEQ\_PARAMETERS**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xapofx.fxeq_parameters(v=vs.85)) | [**FXEQ Constants**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee417670(v=vs.85)) | Only supports FLOAT32 audio formats. The sample rate must be between 22,000 Hz and 48,000 Hz. |
| FXMasteringLimiter | A volume limiter. | [**FXMASTERINGLIMITER\_PARAMETERS**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xapofx.fxmasteringlimiter_parameters(v=vs.85)) | [**FXMASTERINGLIMIT Constants**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee417672(v=vs.85)) | Only supports FLOAT32 audio formats. |
| FXReverb | A simple reverb effect.  XAudio2 also provides an effect implementing Princeton Digital Reverb that can be instantiated with [**XAudio2CreateReverb**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xaudio2.xaudio2createreverb(v=vs.85)). | [**FXREVERB\_PARAMETERS**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xapofx.fxreverb_parameters(v=vs.85)) | [**FXREVERB Constants**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee417674(v=vs.85)) | Only supports FLOAT32 audio formats. Also, it only supports mono input to mono output, and stereo input to stereo output. |

## Creating an Instance of an Effect Included in XAPOFX

XAPOFX provides the [**CreateFX**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/hh405044(v=vs.85)) function as a common mechanism for creating effect instances. **CreateFX** takes the CLSID of an effect, and returns an IUnknown interface pointer to an instance of the effect.

## Using XAPOFX in XAudio2

Effects instantiated with [**CreateFX**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/hh405044(v=vs.85)) are used in XAudio2 by attaching them to voices. Each XAudio2 voice has an effect chain containing zero or more audio effects. Audio data sent to a voice is passed through each effect in the chain before it is sent to the voice's output targets. The voice takes the output of each effect, and feeds it into the next effect in the chain until no effects are left in the chain. To attach an XAPOFX effect to an XAudio2 voice, fill out an [**XAUDIO2\_EFFECT\_CHAIN**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xaudio2.xaudio2_effect_chain(v=vs.85)) structure with the effect's information, and pass it to [**IXAudio2Voice::SetEffectChain**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voice.ixaudio2voice.seteffectchain(v=vs.85)).

For more information about XAudio2 effect chains, see [XAudio2 Audio Effects](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415756(v=vs.85)).

For an example of using XAPOFX in XAudio2, see [How to: Use XAPOFX in XAudio2](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415723(v=vs.85)).

## XAudio2 Implicit Effects

In addition to the library of XAPOs provided by XAPOFX, XAudio2 has built-in reverb and volume meter audio effects. You can create these built-in effects with [**XAudio2CreateReverb**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xaudio2.xaudio2createreverb(v=vs.85)) and [**XAudio2CreateVolumeMeter**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xaudio2.xaudio2createvolumemeter(v=vs.85)). See [How to: Create an Effect Chain](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415789(v=vs.85)) for an example of using one of these built-in effects.

How to: Use XAPOFX in XAudio2

This topic shows you how to use one of the effects included in XAPOFX in an XAudio2 effect chain.

To use an effect from XAPOFX in an XAudio2 effect chain

1. Create the effect by passing the CLSID of an XAPOFX effect to the [**CreateFX**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/hh405044(v=vs.85)) function.

In this case, the simplified reverb effect FXReverb is being created.

IUnknown \* pXAPO;

CreateFX(\_\_uuidof(FXReverb),&pXAPO);

1. Populate an [**XAUDIO2\_EFFECT\_DESCRIPTOR**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xaudio2.xaudio2_effect_descriptor(v=vs.85)) structure with data.
2. XAUDIO2\_EFFECT\_DESCRIPTOR descriptor;
3. descriptor.InitialState = true;
4. descriptor.OutputChannels = 1;
5. descriptor.pEffect = pXAPO;
6. Populate an [**XAUDIO2\_EFFECT\_CHAIN**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xaudio2.xaudio2_effect_chain(v=vs.85)) structure with data.
7. XAUDIO2\_EFFECT\_CHAIN chain;
8. chain.EffectCount = 1;
9. chain.pEffectDescriptors = &descriptor;
10. Apply the effect chain to an XAudio2 voice with the [**SetEffectChain**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voice.ixaudio2voice.seteffectchain(v=vs.85)) function.
11. pVoice->SetEffectChain(&chain);

**Note**  You can also apply an effect chain to a voice when you create the voice by passing the chain as a parameter to[**IXAudio2::CreateSourceVoice**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2.ixaudio2.createsourcevoice(v=vs.85)), [**IXAudio2::CreateSubmixVoice**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2.ixaudio2.createsubmixvoice(v=vs.85)), or [**IXAudio2::CreateMasteringVoice**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/hh405048(v=vs.85)).

1. Release the effect with IUnknown::Release. When you create an XAPO, it will have a reference count of 1. When the XAPO is passed to XAudio2 with[**SetEffectChain**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voice.ixaudio2voice.seteffectchain(v=vs.85)), XAudio2 increments the reference count on the XAPO. Releasing the client's reference to the XAPO allows XAudio2 to take ownership of the XAPO. If XAudio2 has the only reference to the XAPO, this reference is disposed of when it is no longer being used by XAudio2. If the client code needs to maintain a reference to the XAPO—for example, for reuse later—you can skip this step.
2. pXAPO->Release();
3. Populate the parameter structure, if any, associated with the effect.

In this case, the [**FXREVERB\_PARAMETERS**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xapofx.fxreverb_parameters(v=vs.85)) structure is used to set the diffusion and room size that the reverb effect should use.

FXREVERB\_PARAMETERS XAPOParameters;

XAPOParameters.Diffusion = FXREVERB\_DEFAULT\_DIFFUSION;

XAPOParameters.RoomSize = FXREVERB\_DEFAULT\_ROOMSIZE;

1. Pass the effect parameter structure to the effect by calling the [**SetEffectParameters**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voice.ixaudio2voice.seteffectparameters(v=vs.85)) function on the voice to which the effect is attached.
2. hr = pVoice->SetEffectParameters( 0, &XAPOParameters, sizeof( FXREVERB\_PARAMETERS ) );

# Streaming Audio Data

This section provides an overview of XAudio2 streaming.

[XAudio2 Streaming Audio Data](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415821(v=vs.85))

Introduces XAudio2 streaming.

[How to: Stream a Sound from Disk](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415791(v=vs.85))

Describes how to stream audio data from disk with XAudio2.

# XAudio2 Streaming Audio Data

Streaming is the process of maintaining only a small portion of a playing audio file in memory. This allows large audio files such as background music to be played, and not take up a large amount of memory.

When an audio file is streamed, its data is read from disk in chunks rather than loading the entire file at once. Streaming is accomplished by asynchronously reading audio data into a queue of disk buffers. Each buffer is filled, and then submitted to a source voice. After the voice finishes playing a buffer, the buffer becomes available for reading again. Looping through the disk buffers in this manner allows a large audio file to be played while only a portion of its data is loaded. The streaming code should be placed in a separate thread, where it can sleep while waiting for long-running disk and audio operations to finish. A callback class is used to wake the thread by triggering events when audio operations have finished.

For an example of how streaming can be accomplished with XAudio2, see [How to: Stream a Sound from Disk](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415791(v=vs.85)).

# How to: Stream a Sound from Disk

**Note**  This content applies only to desktop apps and will require revision to function in a Windows Store app. Please refer to the documentation for**CreateFile2**, [CreateEventEx](https://msdn.microsoft.com/en-us/library/ms682400(vs.85).aspx), [WaitForSingleObjectEx](https://msdn.microsoft.com/en-us/library/ms687036(vs.85).aspx), [SetFilePointerEx](https://msdn.microsoft.com/en-us/library/aa365542(vs.85).aspx), and **GetOverlappedResultEx**. See the StreamEffect Windows 8 sample from the [Windows SDK Samples Gallery](http://code.msdn.microsoft.com/).

You can stream audio data in XAudio2 by creating a separate thread and perform buffer reads of the audio data in the streaming thread, and then use callbacks to control that thread.

* [Performing buffer reads in the streaming thread](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415791(v=vs.85)#Performing_buffer_reads_in_the_streaming_thread)
* [Creating the callback class](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415791(v=vs.85)#Creating_the_callback_class)

## Performing buffer reads in the streaming thread

To perform buffer reads in the streaming thread follow these steps:

1. Create an array of read buffers.
2. #define STREAMING\_BUFFER\_SIZE 65536
3. #define MAX\_BUFFER\_COUNT 3
4. BYTE buffers[MAX\_BUFFER\_COUNT][STREAMING\_BUFFER\_SIZE];
5. Initialize an OVERLAPPED structure.

The structure is used to check when an asynchronous disk read has finished.

OVERLAPPED Overlapped = {0};

Overlapped.hEvent = CreateEvent( NULL, TRUE, FALSE, NULL );

1. Call the [**Start**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2sourcevoice.ixaudio2sourcevoice.start(v=vs.85)) function on the [**source voice**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2sourcevoice.ixaudio2sourcevoice(v=vs.85)) that will be playing the streaming audio.
2. hr = pSourceVoice->Start( 0, 0 );
3. Loop while the current read position is not passed the end of the audio file.
4. CurrentDiskReadBuffer = 0;
5. CurrentPosition = 0;
6. while ( CurrentPosition < cbWaveSize )
7. {
8. ...
9. }

In the loop, do the following:

* 1. Read a chunk of data from the disk into the current read buffer.
  2. DWORD dwRead;
  3. if( SUCCEEDED(hr) && 0 == ReadFile( hFile, pData, dwDataSize, &dwRead, pOverlapped ) )
  4. hr = HRESULT\_FROM\_WIN32( GetLastError() );
  5. DWORD cbValid = min( STREAMING\_BUFFER\_SIZE, cbWaveSize - CurrentPosition );
  6. DWORD dwRead;
  7. if( 0 == ReadFile( hFile, buffers[CurrentDiskReadBuffer], STREAMING\_BUFFER\_SIZE, &dwRead, &Overlapped ) )
  8. hr = HRESULT\_FROM\_WIN32( GetLastError() );
  9. Overlapped.Offset += cbValid;
  10. //update the file position to where it will be once the read finishes
  11. CurrentPosition += cbValid;
  12. Use the **GetOverlappedResult** function to wait for the event that signals the read has finished.
  13. DWORD NumberBytesTransferred;
  14. ::GetOverlappedResult(hFile,&Overlapped,&NumberBytesTransferred, TRUE);
  15. Wait for the number of buffers queued on the [**source voice**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2sourcevoice.ixaudio2sourcevoice(v=vs.85)) to be less than the number of read buffers.

The state of the [**source voice**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2sourcevoice.ixaudio2sourcevoice(v=vs.85)) is checked with the [**GetState**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/hh405047(v=vs.85)) function.

XAUDIO2\_VOICE\_STATE state;

while( pSourceVoice->GetState( &state ), state.BuffersQueued >= MAX\_BUFFER\_COUNT - 1)

{

WaitForSingleObject( Context.hBufferEndEvent, INFINITE );

}

* 1. Submit the current read buffer to the [**source voice**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2sourcevoice.ixaudio2sourcevoice(v=vs.85)) using the [**SubmitSourceBuffer**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2sourcevoice.ixaudio2sourcevoice.submitsourcebuffer(v=vs.85)) function.
  2. XAUDIO2\_BUFFER buf = {0};
  3. buf.AudioBytes = cbValid;
  4. buf.pAudioData = buffers[CurrentDiskReadBuffer];
  5. if( CurrentPosition >= cbWaveSize )
  6. {
  7. buf.Flags = XAUDIO2\_END\_OF\_STREAM;
  8. }
  9. pSourceVoice->SubmitSourceBuffer( &buf );
  10. Set the current read buffer index to the next buffer.
  11. CurrentDiskReadBuffer++;
  12. CurrentDiskReadBuffer %= MAX\_BUFFER\_COUNT;

1. After the loop has finished, wait for the remaining queued buffers to finish playing.

When the remaining buffers have finished playing, the sound stops, and the thread can exit or be reused to stream another sound.

XAUDIO2\_VOICE\_STATE state;

while( pSourceVoice->GetState( &state ), state.BuffersQueued > 0 )

{

WaitForSingleObjectEx( Context.hBufferEndEvent, INFINITE, TRUE );

}

## Creating the callback class

To create the callback class, create a class that inherits from the [**IXAudio2VoiceCallback**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voicecallback.ixaudio2voicecallback(v=vs.85)) interface.

The class should set an event in its [**OnBufferEnd**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voicecallback.ixaudio2voicecallback.onbufferend(v=vs.85)) method. This allows the streaming thread to put itself to sleep until the event signals it that XAudio2 has finished reading from an audio buffer. For more information about using callbacks with XAudio2, see [How to: Use Source Voice Callbacks](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415769(v=vs.85)).

struct StreamingVoiceContext : public IXAudio2VoiceCallback

{

HANDLE hBufferEndEvent;

StreamingVoiceContext(): hBufferEndEvent( CreateEvent( NULL, FALSE, FALSE, NULL ) ){}

~StreamingVoiceContext(){ CloseHandle( hBufferEndEvent ); }

void OnBufferEnd( void\* ){ SetEvent( hBufferEndEvent ); }

...

};

# X3DAudio

X3DAudio is an API used in conjunction with [XAudio2](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415737(v=vs.85)) to create the illusion of a sound coming from a point in 3D space.

## In this section

|  |  |
| --- | --- |
| **Topic** | **Description** |
| [X3DAudio Overview](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415717(v=vs.85)) | X3DAudio is an API used with [XAudio2](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415737(v=vs.85)) to position sound in 3D space to create the illusion of sound coming from a point in space relative to the position of the camera. |
| [How to: Integrate X3DAudio with XAudio2](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415798(v=vs.85)) | This topic shows how to integrate X3DAudio with XAudio2. |

# X3DAudio Overview

X3DAudio is an API used with [XAudio2](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415737(v=vs.85)) to position sound in 3D space to create the illusion of sound coming from a point in space relative to the position of the camera.In particular, titles that feature 3D scenes will want to use X3DAudio. Sounds not requiring 3D positioning, such as soundtracks or non-positioned ambient sounds, may bypass X3DAudio completely.

## Listeners and Emitters

To manage sounds in 3D space, X3DAudio employs the concepts of listeners and emitters. Listeners and emitters represent the position of whatever is hearing 3D sounds, and the point from which those sounds originate.

* A listener is defined as a point in space and an orientation. It is the position at which the sound is heard. The position and orientation of the listener generally is the same as the position and orientation of the camera. This is true whether a title uses a first-person or third-person perspective view. The listener's position is expressed in world coordinates. It is important to note that it is the listener's position relative to an emitter that determines how to calculate the final speaker volumes.
* An emitter is defined as one (or more) points in space from which a sound originates. The position of the emitter can be anywhere in 3D space. Like a listener, an emitter's position is expressed in world coordinates. It is the emitter's position relative to the listener that determines how the final speaker volumes are calculated.
* X3DAudio uses left-handed coordinates. To use with right-handed coordinates, developers need to negate the .z element of the OrientTop, OrientFront, Position, and Velocity members of X3DAUDIO\_LISTENER and X3DAUDIO\_EMITTER.

In addition to position, listeners and emitters can include velocity. Unlike a 3D rendering engine, X3DAudio only uses velocity to calculate Doppler effects (it is not used to calculate position).

For more details about listeners and emitters, see the [**X3DAUDIO\_LISTENER**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.x3daudio.x3daudio_listener(v=vs.85)) and [**X3DAUDIO\_EMITTER**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.x3daudio.x3daudio_emitter(v=vs.85)) structure reference topics.

## Using X3DAudio with XAudio2

For all interaction between X3DAudio and XAudio2, use the following X3DAudio functions.

* [**X3DAudioInitialize**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.x3daudio.x3daudioinitialize(v=vs.85))

Call the [**X3DAudioInitialize**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.x3daudio.x3daudioinitialize(v=vs.85)) function to initialize X3DAudio. Typically, you only need to call **X3DAudioInitialize** once in the lifetime of a game, unless the speaker configuration is changed.

* [**X3DAudioCalculate**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.x3daudio.x3daudiocalculate(v=vs.85))

After you initialize X3DAudio, you can determine volume and other values for a given sound by passing the sound's emitter and the listener to the[**X3DAudioCalculate**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.x3daudio.x3daudiocalculate(v=vs.85)) function. The values calculated by **X3DAudioCalculate** can then be applied to XAudio2 voices or effects as appropriate for the flags passed to the function. You can apply volume and pitch values calculated by X3DAudio to a voice with the [**IXAudio2Voice::SetOutputMatrix**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voice.ixaudio2voice.setoutputmatrix(v=vs.85)) and[**IXAudio2SourceVoice::SetFrequencyRatio**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2sourcevoice.ixaudio2sourcevoice.setfrequencyratio(v=vs.85)) methods. Other values calculated by X3DAudio will need to be applied to a [**reverb effect**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xaudio2.xaudio2createreverb(v=vs.85)) using the[**IXAudio2Voice::SetEffectParameters**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voice.ixaudio2voice.seteffectparameters(v=vs.85)) method.

# How to: Integrate X3DAudio with XAudio2

This topic shows how to integrate X3DAudio with XAudio2. You can use X3DAudio to provide the volume and pitch values for XAudio2 voices and the parameters for the XAudio2 built in reverb effect. This topic assumes that you have created an audio graph as described in [How to: Build a Basic Audio Processing Graph](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415767(v=vs.85)). If you have not already created an audio graph, [**X3DAudioInitialize**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.x3daudio.x3daudioinitialize(v=vs.85)) will fail.

Ee415798.wedge(en-us,VS.85).gif**To initialize X3DAudio**

1. Initialize X3DAudio by calling [**X3DAudioInitialize**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.x3daudio.x3daudioinitialize(v=vs.85)).

The [**X3DAudioInitialize**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.x3daudio.x3daudioinitialize(v=vs.85)) function takes flags indicating the speaker setup, the speed of sound in user-defined world units per second, and a handle to return an instance of the X3DAudio engine. Call [**IXAudio2MasteringVoice::GetChannelMask**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2masteringvoice.ixaudio2masteringvoice.getchannelmask(v=vs.85)) to get the output format's channel mask.

DWORD dwChannelMask;

pMasteringVoice->GetChannelMask( &dwChannelMask );

X3DAUDIO\_HANDLE X3DInstance;

X3DAudioInitialize( dwChannelMask, X3DAUDIO\_SPEED\_OF\_SOUND, X3DInstance );

1. Create instances of the [**X3DAUDIO\_LISTENER**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.x3daudio.x3daudio_listener(v=vs.85)) and [**X3DAUDIO\_EMITTER**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.x3daudio.x3daudio_emitter(v=vs.85)) structures.

The [**X3DAUDIO\_LISTENER**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.x3daudio.x3daudio_listener(v=vs.85)) structure represents the position of whatever is hearing the sound. Generally, this is the position of the camera or a position close to it. The [**X3DAUDIO\_EMITTER**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.x3daudio.x3daudio_emitter(v=vs.85)) structure represents the position of the thing making the sound. There will be one **X3DAUDIO\_EMITTER**structure for each sound that is being tracked.

Members of the structures that will not be updated in a game loop should be initialized here. Most members of the structures can simply be initialized to zero. However, some members of [**X3DAUDIO\_EMITTER**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.x3daudio.x3daudio_emitter(v=vs.85)) need to be set to be initialized to non-zero values. The ChannelCount member of the**X3DAUDIO\_EMITTER** needs to be initialized to the number of channels in the voice the emitter represents. Also, the CurveDistanceScaler member of**X3DAUDIO\_EMITTER** must be in the range FLT\_MIN to FLT\_MAX.

X3DAUDIO\_LISTENER Listener = {0};

X3DAUDIO\_EMITTER Emitter = {0};

Emitter.ChannelCount = 1;

Emitter.CurveDistanceScaler = FLT\_MIN;

1. Create an instance of the [**X3DAUDIO\_DSP\_SETTINGS**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.x3daudio.x3daudio_dsp_settings(v=vs.85)) structure.

The [**X3DAUDIO\_DSP\_SETTINGS**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.x3daudio.x3daudio_dsp_settings(v=vs.85)) structure is used to return results calculated by [**X3DAudioCalculate**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.x3daudio.x3daudiocalculate(v=vs.85)). The **X3DAudioCalculate** function does not allocate memory for any of its parameters. This means that you need to allocate arrays for the **X3DAUDIO\_DSP\_SETTINGS** structure's pMatrixCoefficients and pDelayTimes members if you intend to use them. In addition, you need to set the SrcChannelCount and DstChannelCount members to the number of channels in the emitter's source and destination voices.

X3DAUDIO\_DSP\_SETTINGS DSPSettings = {0};

FLOAT32 \* matrix = new FLOAT32[deviceDetails.OutputFormat.Format.nChannels];

DSPSettings.SrcChannelCount = 1;

DSPSettings.DstChannelCount = deviceDetails.OutputFormat.Format.nChannels;

DSPSettings.pMatrixCoefficients = matrix;

**Note**  Use [**IXAudio2Voice::GetVoiceDetails**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voice.ixaudio2voice.getvoicedetails(v=vs.85)) on the mastering voice to obtain the number of InputChannels for **nChannels**. For DirectX SDK versions of XAUDIO2 prior to Windows 8, use IXAudio2::GetDeviceDetails.

Perform these steps once every two to three frames to calculate new settings and apply them. In this example, a source voice is sending directly to the mastering voice and to a submix voice with a reverb effect applied to it.

Ee415798.wedge(en-us,VS.85).gif**To use X3DAudio to calculate and apply new 3D audio settings**

1. Update the [**X3DAUDIO\_LISTENER**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.x3daudio.x3daudio_listener(v=vs.85)) and [**X3DAUDIO\_EMITTER**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.x3daudio.x3daudio_emitter(v=vs.85)) structures with their current position, velocity, and orientation.
2. Emitter.OrientFront = EmitterOrientFront;
3. Emitter.OrientTop = EmitterOrientTop;
4. Emitter.Position = EmitterPosition;
5. Emitter.Velocity = EmitterVelocity;
6. Listener.OrientFront = ListenerOrientFront;
7. Listener.OrientTop = ListenerOrientTop;
8. Listener.Position = ListenerPosition;
9. Listener.Velocity = ListenerVelocity;
10. Call [**X3DAudioCalculate**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.x3daudio.x3daudiocalculate(v=vs.85)) to calculate new settings for the voices.

The parameters for [**X3DAudioCalculate**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.x3daudio.x3daudiocalculate(v=vs.85)) will be the updated [**X3DAUDIO\_LISTENER**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.x3daudio.x3daudio_listener(v=vs.85)) and [**X3DAUDIO\_EMITTER**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.x3daudio.x3daudio_emitter(v=vs.85)) structures. The flags will indicate what values **X3DAudioCalculate** should calculate, and which [**X3DAUDIO\_DSP\_SETTINGS**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.x3daudio.x3daudio_dsp_settings(v=vs.85)) structure will hold the results of the calculations performed.

X3DAudioCalculate(X3DInstance, &Listener, &Emitter,

X3DAUDIO\_CALCULATE\_MATRIX | X3DAUDIO\_CALCULATE\_DOPPLER | X3DAUDIO\_CALCULATE\_LPF\_DIRECT | X3DAUDIO\_CALCULATE\_REVERB,

&DSPSettings );

1. Use [**IXAudio2Voice::SetOutputMatrix**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voice.ixaudio2voice.setoutputmatrix(v=vs.85)) and [**IXAudio2SourceVoice::SetFrequencyRatio**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2sourcevoice.ixaudio2sourcevoice.setfrequencyratio(v=vs.85)) to apply the volume and pitch values to the source voice.
2. pSFXSourceVoice->SetOutputMatrix( pMasterVoice, 1, deviceDetails.OutputFormat.Format.nChannels, DSPSettings.pMatrixCoefficients ) ;
3. pSFXSourceVoice->SetFrequencyRatio(DSPSettings.DopplerFactor);
4. Use [**IXAudio2Voice::SetOutputMatrix**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voice.ixaudio2voice.setoutputmatrix(v=vs.85)) to apply the calculated reverb level to the submix voice.
5. pSFXSourceVoice->SetOutputMatrix(pSubmixVoice, 1, 1, &DSPSettings.ReverbLevel);
6. Use [**IXAudio2Voice::SetFilterParameters**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voice.ixaudio2voice.setfilterparameters(v=vs.85)) to apply the calculated low pass filter direct coefficient to the source voice.
7. XAUDIO2\_FILTER\_PARAMETERS FilterParameters = { LowPassFilter, 2.0f \* sinf(X3DAUDIO\_PI/6.0f \* DSPSettings.LPFDirectCoefficient), 1.0f };
8. pSFXSourceVoice->SetFilterParameters(&FilterParameters);

# XAudio2 Operation Sets

This overview introduces several XAudio2 methods that you can call as part of an operation set.

Several XAudio2 methods take the OperationSet argument, which allows them to be called as part of a deferred group. At a specific time, you can apply an entire set of changes simultaneously by calling the function [**IXAudio2::CommitChanges**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2.ixaudio2.commitchanges(v=vs.85)) with the OperationSet identifier for that group. The identifier is an arbitrary number. Thus, it allows separate parts of the client code to apply separate atomic changes to the graph without any conflict. The recommended practice is for the client to increment a global counter whenever it needs to generate a unique, new OperationSet identifier. A set of changes to the graph, applied atomically, is guaranteed to be sample-accurate. For example, voices will start in sync.

If you set OperationSet to XAUDIO2\_COMMIT\_NOW, the change applies immediately. It takes effect in the first audio processing pass after the method call. If you call [**CommitChanges**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2.ixaudio2.commitchanges(v=vs.85)) with XAUDIO2\_COMMIT\_ALL, changes to all pending operations sets are performed, regardless of their OperationSetidentifier.

Certain methods take effect immediately when they are called from an XAudio2 callback with an OperationSet of XAUDIO2\_COMMIT\_NOW. All other methods that take an OperationSet argument only take effect on the next processing pass after the method is called (if called with XAUDIO2\_COMMIT\_NOW), or after [**CommitChanges**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2.ixaudio2.commitchanges(v=vs.85)) is called with the same OperationSet. Because of this, certain method calls may not always happen in the same order in which they were called.

All pending operations are committed atomically when [**IXAudio2::StopEngine**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2.ixaudio2.stopengine(v=vs.85)) is called. Any methods that are called while the engine is stopped take effect immediately, regardless of the OperationSet value provided. When you restart the engine, XAudio2 returns to asynchronous mode.

Simple scenarios in which operation sets are useful include the following examples.

* Starting multiple voices simultaneously.
* Simultaneously submitting a buffer to a voice, setting the voice parameters, and starting the voice.
* Making a large-scale change to the graph, such as connecting all source voices to a new submix voice.

See [How to: Group Audio Methods as an Operation Set](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415783(v=vs.85)) for an example of using an operation set.

## Operation Set Methods

You can call the following methods as part of an operation set.

* [**IXAudio2SourceVoice::ExitLoop**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2sourcevoice.ixaudio2sourcevoice.exitloop(v=vs.85))
* [**IXAudio2Voice::SetFilterParameters**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voice.ixaudio2voice.setfilterparameters(v=vs.85))
* [**IXAudio2SourceVoice::SetFrequencyRatio**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2sourcevoice.ixaudio2sourcevoice.setfrequencyratio(v=vs.85))
* [**IXAudio2Voice::DisableEffect**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voice.ixaudio2voice.disableeffect(v=vs.85))
* [**IXAudio2Voice::EnableEffect**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voice.ixaudio2voice.enableeffect(v=vs.85))
* [**IXAudio2Voice::SetChannelVolumes**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voice.ixaudio2voice.setchannelvolumes(v=vs.85))
* [**IXAudio2Voice::SetEffectParameters**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voice.ixaudio2voice.seteffectparameters(v=vs.85))
* [**IXAudio2Voice::SetOutputMatrix**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voice.ixaudio2voice.setoutputmatrix(v=vs.85))
* [**IXAudio2Voice::SetVolume**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2voice.ixaudio2voice.setvolume(v=vs.85))
* [**IXAudio2SourceVoice::Start**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2sourcevoice.ixaudio2sourcevoice.start(v=vs.85))
* [**IXAudio2SourceVoice::Stop**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2sourcevoice.ixaudio2sourcevoice.stop(v=vs.85))

As described previously, client code must ultimately call the function [**IXAudio2::CommitChanges**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2.ixaudio2.commitchanges(v=vs.85)) to execute the deferred changes.

# Debugging Facilities

This section lists the topics about XAudio2 debugging facilities.

## In This Section

|  |  |
| --- | --- |
| **Term** | **Description** |
| [XAudio2 Debugging Facilities](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415752(v=vs.85)) | Introduces the XAudio2 debugging facilities. |
| [Debugging Audio Glitches in XAudio2](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415765(v=vs.85)) | Describes ways to diagnose and avoid audio glitches in XAudio2. |

XAudio2 Debugging Facilities

The debug version of the XAudio2 engine validates parameters, and provides detailed warning and error messages.

Setting the Debug Logging Level at Run Time

You can set the level of debugging information shown by XAudio2 at any time by filling out an [**XAUDIO2\_DEBUG\_CONFIGURATION**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xaudio2.xaudio2_debug_configuration(v=vs.85)) structure with the flags for the desired logging level, and then pass the structure to the [**IXAudio2::SetDebugConfiguration**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2.ixaudio2.setdebugconfiguration(v=vs.85)) method. Values passed to the**IXAudio2::SetDebugConfiguration** method always override any default values that were set in the Windows registry.

Debug Support

The debugging facilities are always availlabe for XAUDIO2 in Windows 8.

For the DirectX SDK versions of XAUDIO2, you must use **XAUDIO2\_DEBUG\_ENGINE** when creating the XAUDIO2 object with [**XAudio2Create**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xaudio2.xaudio2create(v=vs.85)) and the system must have the DirectX SDK Developer Runtime installed for debugging to be supported.

# Debugging Audio Glitches in XAudio2

Glitches can occur in XAudio2, this topic covers how they are reported and some approaches to fixing them.

This overview covers the following topics:

* [Causes of audio output problems or glitches](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415765(v=vs.85)#causes_of_audio_output_problems__or_glitches)
* [How XAudio2 reports problems](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415765(v=vs.85)#how_xaudio2_reports_problems)
* [Approaches to fixing problems](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415765(v=vs.85)#approaches_to_fixing_problems)
* [Related topics](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415765(v=vs.85)#related_topics)

## Causes of audio output problems or glitches

Glitches can occur in XAudio2 output for several reasons.

* An XAudio2 source voice is starved. The client is not submitting fresh audio to it fast enough. You get silence because it has no data to play.
* XAudio2 as a whole is overburdened. It takes longer than X ms to produce X ms of audio. You get dropouts because XAudio2 can't produce data as fast as the audio device needs it. You might be running too many voices or effects at a time, doing too much work in XAudio2 callbacks, or making XAudio2 API calls too frequently.
* The audio processing thread is stalling because the client's implementation of some XAudio2 callback is doing things that can block the thread. For example, it might be accessing the disk, synchronizing with other threads, or calling other functions that may block. Use a lower-priority background thread that the callback can signal to perform such tasks.
* The system as a whole is overloaded. Other threads running at the same or higher priority than XAudio2 are doing too much work. They are competing with the audio thread for CPU time.

## How XAudio2 reports problems

XAudio2 can communicate glitches in the debug build in several ways.

* If a voice is being starved, XAudio2 shows a message in this form.
* XAudio2: WARNING: Voice at 0xNNNNNNNN starved: no more source buffers are available, but no end-of-stream marker was received
* If the audio thread runs for too long, XAudio2 shows a message in this form.
* XAudio2: WARNING: Spent Xms in audio thread; XAudio2 possibly overloaded

Typically, this message occurs with the next message.

* If the audio driver can't be fed new audio data on time, XAudio2 shows a message in this form.
* XAudio2: WARNING: Glitch at output sample X
* Calling [**IXAudio2::GetPerformanceData**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2.ixaudio2.getperformancedata(v=vs.85)) provides XAudio2 performance data, including the total number of glitches since the XAudio2 engine started.

## Approaches to fixing problems

Possible ways to reduce audio glitches include the following.

* In the voice starvation case: Increase the amount of audio data that is queued ahead on a voice. You can use [**IXAudio2SourceVoice::GetState**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/hh405047(v=vs.85)) to discover the number of buffers queued at any moment. If you still see voice starvation errors, but can't hear any glitch, make sure you are setting[**XAUDIO2\_BUFFER**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xaudio2.xaudio2_buffer(v=vs.85)).**Flags** to XAUDIO2\_END\_OF\_STREAM on the final buffer of a sound. This tells XAudio2 not to expect any more buffers necessarily to be available as soon as this one completes.

In the other cases:

* + Reduce the number of active voices and effects in the graph, especially expensive effects like reverb.
  + Disable voices and effects you're not using.
  + Use the XAUDIO2\_VOICE\_NOSRC and XAUDIO2\_VOICE\_NOPITCH flags in [**IXAudio2::CreateSourceVoice**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2.ixaudio2.createsourcevoice(v=vs.85)), whenever possible. Sample rate conversion is costly.
  + Reduce the sample rate of individual voices. For example, a submix voice hosting a reverb effect can have a lower sample rate than the source voice sending to it. Sounds such as explosions and gunshots that don't need high fidelity can also be recorded at lower sample rates.
  + Ensure that callback implementations do as little work as possible and never block.
  + Make fewer calls to XAudio2. Audio parameters usually don't need to be updated for every video frame. Every 30 ms or so is sufficient. You should eliminate redundant calls, such as setting volume several times in quick succession.
  + Reduce the game's overall CPU usage.

# ADPCM Overview

Adaptive Differential Pulse Code Modulation (ADPCM) is a lossy compression format that is implemented for XAudio2 to provide additional features for specifying the size of the compression sample block. With a lossy compression format some data is altered and lost during compression. ADPCM can achieve compression ratios of up to 4:1.

The implementation of ADPCM for XAudio2 provides additional features to specify the size of the compression sample block. ADPCM enables the audio designer to choose a setting that is an appropriate compromise among size, quality, and resolution (for placing loop points).

XAudio2 uses a modified version of the Microsoft ADPCM codec that supports the extended data formatting required to provide custom sample block sizes. For this reason, XAudio2 audio data cannot be played by audio engines that do not support this version of the ADPCM codec.

**Note**  Currently, ADPCM compression is only available for Windows, including XNA Game Studio Express for Windows deployments.

## ADPCM Encoding

Audio data is encoded to ADPCM using the AdpcmEncode command-line tool.

* AdpcmEncode

In order to encode audio files as ADPCM for use with XAudio2, use the **AdpcmEncode** command-line tool.

## ADPCM Decoding

Software decoding of ADPCM is supported in XAudio2.

* XAudio2

In order to use ADPCM encoded data in XAudio2, you need to initialize a **ADPCMWAVEFORMAT** structure with ADPCM specific values, and pass it as an argument to [**IXAudio2::CreateSourceVoice**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.ixaudio2.ixaudio2.createsourcevoice(v=vs.85)) when you create a source voice. For an example of loading and playing a sound in XAudio2, see [How to: Play a Sound with XAudio2](https://msdn.microsoft.com/zh-cn/library/windows/desktop/ee415787(v=vs.85)).

## SamplesPerBlock

ADPCM compression works by separating the waveform into blocks, and predicting the variation of the waveform samples within each block. The size of the blocks is measured in samples. The smallest block size is 32 samples, and the highest is 512 samples.

Larger blocks allow better compression, which results in smaller file sizes, but at the expense of sound quality and resolution for aligning loop points.

In general, modifying the SamplesPerBlock value results in these tradeoffs:

|  |  |  |  |
| --- | --- | --- | --- |
| **If SamplesPerBlock...** | **File Compression** | **Sound Quality** | **Loop Point Resolution** |
| Increases (up to max 512) | Increases | Decreases | Decreases |
| Decreases (down to min 32) | Decreases | Increases | Increases |

## Restrictions

Because ADPCM uses sample blocks that are aligned one after the other, a wave compressed with ADPCM may have an unfinished, partial block at its end. The ADPCM decoder generates silence for the remainder of this partial block, which keeps the wave from looping seamlessly.

The value of the SamplesPerBlock parameter affects the resolution with which you can align wave data and loop points.

If you try to apply compression to a non-aligned wave, you will get an error or a warning depending on whether the wave is used in any looping play events. You cannot compress a wave used in any looping play events. Remove it from the looping play events, and re-apply compression.

If you use the wave exclusively in non-looping mode, the sample block alignment restriction does not apply.

## ADPCM File Structure

An ADPCM file is a standard RIFF file with the following chunk types.

|  |  |
| --- | --- |
| **Chunk FCC** | **Description** |
| RIFF | Standard RIFF chunk containing a file type with the value WAVE in the first four bytes of its data section and the other chunks in the file in the remainder of its data section. |
| fmt | Contains the format header for the ADPCM file. The data in this chunk corresponds to a **ADPCMWAVEFORMAT** structure. |
| data | Contains the encoded ADPCM audio data. When you use ADPCM in XAudio2, you need to read the contents of the data chunk into a buffer, and pass it to a source voice as the **pAudioData** member of an [**XAUDIO2\_BUFFER**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xaudio2.xaudio2_buffer(v=vs.85)) structure. You don't need to byte swap the contents of the data chunk. |
| smpl and wsmp | Optional chunk types containing the looping information for the ADPCM file. When you use ADPCM in XAudio2, the values contained in the smpl or wsmp chunks are used to populate the **LoopBeginLoopLength** and **LoopCount** members of the [**XAUDIO2\_BUFFER**](https://msdn.microsoft.com/zh-cn/library/windows/desktop/microsoft.directx_sdk.xaudio2.xaudio2_buffer(v=vs.85))structure. On the Xbox 360, you need to byte swap the data loaded from a smpl chunk to account for the endianness difference between Windows and Xbox 360. |