# Environmental Audio Extensions: EAX 2.0

## Table of Contents

Version 1.3

## Trademarks and Service Marks

Creative, Sound Blaster, and the Creative logo are registered trademarks, and Environmental Audio, E-mu Environmental Modeling, FourPointSurround, EMU10K1, Creative Multi Speaker Surround, EAX, Environmental Audio Extensions, and the Sound Blaster Live! logo are trademarks of Creative Technology Ltd. in the United States and/or other countries.

E-mu, E-mu Systems, and SoundFont are registered trademarks of E-mu Systems, Inc.

Cambridge Soundworks is a registered trademark and PCWorks is a trademark of Cambridge Soundworks Inc., Newton, MA.

Microsoft, MS-DOS, Windows, DirectSound, DirectX, and DirectSound 3D are registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

All other brand and product names listed are trademarks or registered trademarks of their respective holders.

## Acknowledgments

Documentation written by Jean-Marc Jot and Michael Boom. Additional input by Sam Dicker, Luke Dahl, Keith Charley, and Garin Hiebert.

# Environmental Audio Extensions: EAX 2.0

Version 1.3

Creative®'s Environmental Audio Extensions (EAX™) add 3D reverberation capabilities with occlusion and obstruction effects to the DirectSound™ component of DirectX or to OpenAL. This document describes EAX and its place in DirectSound and OpenAL. It also describes the sound design models in place behind EAX and shows you how to work with them to use EAX in your own code.

## INTRODUCING EAX

A primary goal of computer gaming is to produce a realistic 3D world for the player. Creating the aural component of that world has always lagged behind the visual component. It's not uncommon to find a computer displaying sophisticated 3D graphics with texture mapping, shading, multiple light sources, haze, and more while one or two tinny speakers spit out monaural sound.

### DIRECTSOUND AND OPENAL SOLUTIONS

Microsoft's DirectSound (the aural component of DirectX) or OpenAL provide a major first step for creating a realistic 3D aural world: each creates an easy-to-use programming environment for 3D aural modeling in C or C++. You can use either API to create separate sound sources that move around realistically in the 3D aural world along with their corresponding objects in the 3D visual world: as the berserker warrior falls off the cliff, the player perceives his scream going with him.

Both DirectSound and OpenAL keep the nitty gritty of audio hardware at arm's length from the programmer. The programmer uses the relatively simple API to create sound sources, set their 3D positions and velocities (if moving), and take care of other decisions about the quality and placement of sounds in the aural world. DirectSound or OpenAL, through the audio driver installed in a computer, does the work of translating sound-source waveforms, positions, velocity, and more into a mix that ultimately comes out in realistic 3D form through the player's speakers or headphones.

Although DirectSound and OpenAL provide a number of sophisticated 3D aural effects such as Doppler effect, rolloff, and direction, they lack some very important environmental effects: reverberation, reflections, and sound occlusion or obstruction by intervening objects. Without these environmental effects, a listener can tell the direction of each sound source, but has no idea of the environment where the sources are located. The listener also has a more difficult time telling how far away the sources are.

Consider a sword clanked in a small padded cell. It should sound much different than the same sword clanked in a large cathedral, and it's reverberation that tells the story. Or, consider a scream coming from the next room. The occluded (muffled) quality of the scream tells you there's a wall in between you and the screamer. Without environmental audio effects, sound sources are naked and lack warmth—the aural equivalent of a visual world without shadows, haze, and independent light sources.

## EAX'S SOLUTIONS

In DirectSound, Creative's EAX property sets add reverberation and occlusion/obstruction effects through property set extensions. As DirectSound property sets, they use the DirectSound API and the COM (Component Object Model) used throughout the DirectX environment. You can create an EAX interface for each sound source in the 3D aural world and then set the reverberation amount or control the occlusion and obstruction effects for individual sound sources.

In OpenAL, Creative's EAX is added as an API extension. The 3D aural world is established through OpenAL and EAX properties are attached to sound sources to control reverberation, occlusion, and obstruction effects.

With both DirectSound and OpenAL, you can also control the overall quality of the reverberation the listener hears, tweaking reverberation factors such as:

- the apparent size of the room surrounding the listener
- the duration and tonal color of the reverberation's decay
- the amount and delay of reflections and reverberation.

These effects combine to add a visceral realism to the aural environment, an often-subliminal context that can give an emotional depth to the 3D world of the player. All of this works even when the visual component of the 3D world is out of sight. Think, for example, of a single candle next to a pond of water in dark surroundings. When a drop of water hits the pond and you hear long and luscious reverberation on the plink of the drop, your mind senses the vast cavern surrounding the pond even though you can't see it.

If you don't care for reverberation in some circumstances, you can turn it off on a per-sound-source basis or turn it off altogether.

## 3D Effects

These EAX enhancements create a much more robust perception of individual sound source distance or proximity that establishes the depth of the sound scene. When you move a sound source in relation to the listener, EAX can automatically adjust the reverberation for the sound source (increasing the ratio of reverberation for a source moving away, for example) to simulate the behavior of natural environments. None of the 3D effects are lost in the mix; they're augmented with reverberation calculated to enhance the feeling of three dimensions.

## Environments

EAX provides reverberation environments that make it easy to simulate any one of a large variety of acoustic surroundings. Each EAX environment simulates a given set of acoustic surroundings such as an auditorium, a padded cell, an arena, a stone corridor, underwater, a city street, and so on. All you have to do is specify the environment you want. EAX takes care of the rest, supplying realistic and interactive 3D reverberation and reflections for all the sound sources within the environment you choose. You can, if you wish, tweak the reverberation quality of any environment to get the exact acoustic surroundings you want.

## An Open Standard

EAX is an open standard that takes advantage of any hardware-accelerated card (such as Sound Blaster® Live!) that provides the necessary reverberation and occlusion/obstruction processing. In either DirectSound or OpenAL, when your application first asks for EAXsupport, the card's audio driver is queried to see if EAX support is available. If EAX is not available, the application will still work properly, but will not have EAX effects.

As an open standard, EAX works not only with Creative's cards, but also with any manufacturer's cards that care to take advantage of the EAX property sets.

## WHAT'S NEW IN EAX 2.0

Version 2.0 of EAX adds significant enhancements to version 1.0. These enhancements provide much more detail to the aural world perceived by the listener. They also give programmers more power for sound design by controlling specific environmental audio effects.

### More Environmental Reverberation Control

EAX 2.0 offers a complete set of reverberation parameters that allow you to control the intensity and delay of reflections, to continuously adjust the environment size, and to set the direct path and reverberation filters for each sound source. These direct controls allow dedicated sound designers to tweak environmental reverberation to get precisely the aural surroundings they desire. EAX can also make your programming work easier by automatically taking care of these parameters for you.

### Distance Effects

To help provide a better perception of listener-to-sound-source distance, EAX 2.0 provides several different modes of automatic distance-effects management. They manage attenuation and filtering effects that change according to the distance between the listener and a sound source. The distance effects include:

- A method of simulating the natural statistical behavior of reverberation in enclosed spaces
- A reverberation rolloff factor that allows you to customize or modify the statistical reverberation model
- An adjustable air absorption model that filters out high frequencies with increasing distance. (You can use this to simulate moisture in the air such as high humidity or fog.)

## Occlusion and Obstruction Effects

EAX 2.0 introduces occlusion and obstruction properties that provide aural cues about objects and walls that come between sound sources and the listener.

Occlusion occurs when a wall that separates two environments comes between source and listener. There's no open-air sound path for sound to go from source to listener, so the sound source is completely muffled because it's transmitted through the wall. EAX occlusion properties provide parameters that adjust wall transmission characteristics to simulate different wall materials and thickness. You can, for example, use occlusion properties to make a voice or noise sound very realistically as if it were coming from behind a door or from outside the listener's house.

Obstruction occurs when source and listener are in the same room but there's an object directly between them. There's no direct sound path from source to listener, but the reverberation comes to the listener unaffected. The result is altered direct-path sound with unaltered reverberation. EAX obstruction properties can simulate sound diffraction around the obstacle or sound transmission through the obstacle, which provide rich aural cues about the nature of the obstruction. You can, for example, use obstruction properties to make a voice sound as if it were coming through a thin curtain or from behind a large pillar.

## Source Directivity

DirectSound and OpenAL already offer control of source directivity, where a sound source is stronger in one direction than in all others—like a trumpet, for example, that's strongest in the direction of the bell and attenuated to the sides and behind the bell. DirectSound and OpenAL set the directivity across all frequencies equally, which usually is not the case in the real world.

EAX makes source directivity sound much more natural by allowing you to make sound sources more directive at high frequencies than at low frequencies. This makes the sound of directional sources such as a voice, a loudspeaker, or a horn, sound much more realistic as the listener moves around the source—or as the source rotates or flies by the listener.

# HOW EAX FITS INTO DIRECTSOUND

The first step in learning how to use EAX with DirectSound is to understand its place in the DirectSound environment. Microsoft has extensive documentation explaining DirectSound, so we won't go into too many DirectSound details here. We'll explore only the main concepts you'll need to understand how EAX fits in.

## DIRECTSOUND PROPERTY SETS

If you're unfamiliar with property sets, Microsoft introduced them as an extension of DirectSound in release 5.0 of DirectX. Property sets are intended to provide access to advanced features of sound cards that aren't available through the DirectSound API. In other words, property sets make DirectSound extensible.

As the name suggests, a property set is a collection of properties, each with a setting or settings. An application checks first to see if the audio driver on the computer can support the property set. If it can, the application sets values for one, some, many, or all of the properties to control the effects provided by the property set.

A third-party hardware developer can create its own property set and then support the property set within the audio driver for an audio card. Each property set has a unique identifier, a *GUID* (Globally Unique Identifier), defined by the set creator, meant to be unique around the world for all time.

When an application wants to use a property set, it asks DirectSound for a property-set interface. The application may then query the interface, using the desired property set's GUID, to see if the sound card supports the property set. If the set is supported, the application can set a property by specifying the property set (by GUID), a property within that set (by number), and a value or array of values for that property. The application can also get a property's current setting by specifying GUID and property number.

## PRIMARY AND SECONDARY SOUND BUFFERS

Primary and secondary sound buffers are important parts of the DirectSound environment. They provide a mechanism for creating sound sources and a listener in a 3D aural world.

### Sound Buffers

Sound buffers are used throughout DirectSound to contain a waveform table, a set of values that—converted to an analog signal—produce sound on an audio system. The application using DirectSound is responsible for providing the waveform data for source sound buffers and then telling DirectSound how to play back that waveform data. An application can synthesize the waveform data for a sound buffer, insert a sound sample in the buffer, or stream audio to the buffer for playback.

### The Primary Sound Buffer (The Listener)

DirectSound always has a single primary buffer—never more, never less—to feed waveform data directly to the audio system through a DAC (Digital-to-Audio Converter). Although the primary buffer provides only a single waveform table, it can support mono, stereo, or multichannel playback by interleaving samples for each channel within a single table.

Because the primary buffer is the direct waveform feed to the outside world and the player, it represents the *listener* in the 3D aural world, and is often referred to as such. DirectSound can (if set to simulate a 3D environment) assign 3D settings to the primary buffer such as: a location, the velocity, the listener's orientation (looking up, down, left, or right), and so on.

When an audio application runs, the primary buffer receives a mix of waveform data from other sound sources in the 3D aural world. DirectSound keeps track of the location of the other sound sources in relation to the listener and alters their output to simulate three dimensions. It may, for example, reduce a sound source's volume as it increases its distance from the listener, or add a Doppler effect if the source comes whooshing by the listener. An application rarely writes waveform data directly to the primary buffer, which is almost always under the direct control of DirectSound. If an application does take over writing to the primary buffer, it takes over the job of mixing sound sources.

DirectSound allows the application to be ignorant of primary buffer specifics: how and where the buffer is maintained, how the mix occurs. DirectSound handles the details of mixing sound sources, feeding the mix to the primary buffer, and working with the audio card to send the primary buffer contents to the outside world.

## Secondary Sound Buffers (Sound Sources)

DirectSound supports as many secondary sound buffers as the host system is able to accommodate in computer RAM or sound-card RAM. Each of these buffers represents a *sound source.* Each holds a waveform table created by the application for playback and plays back the waveform table as directed by the application. When DirectSound is set to simulate a 3D environment, the application can assign a secondary buffer's position in the 3D aural world as well as its velocity (if it's in motion), its sound cones (for directional sound projection), its minimum-maximum distances from the listener, and so on.

## Objects and Interfaces

Because DirectSound is object oriented, it creates each sound buffer—the primary and any secondary buffers—as objects. To provide application control for each sound object, DirectSound creates a standard interface (also an object) for each sound object and ties the interface to the sound object. An application controls each sound buffer through the buffer's interface by calling member functions on the interface. An application can, for example, call the member function `SetVolume` on a buffer's standard interface to set the buffer's playback volume.

DirectSound gets a little more complicated when you use it to create a 3D aural world: it requires two interfaces for each buffer. The first interface is the standard buffer interface that sets non-3D buffer characteristics such as playback volume or frequency. The second interface is a 3D buffer interface that sets 3D buffer characteristics such as location or velocity.

When you want to use a property set with a 3D buffer, you work with three interfaces: the standard buffer interface for non-3D characteristics, the 3D buffer interface for 3D characteristics, and a property-set interface to set special buffer properties. The property-set interface allows the application to first query to see if a particular property set exists, and then—if it's there—to set property values within the set and to read current property values.

## EAX'S ROLE IN DIRECT SOUND

EAX comprises two different property sets. The first, called the *listener property set*, applies to the primary buffer (the listener) and describes the listener's surrounding environment (the "room" in which the listener listens). The second set, called the *sound-source property set*, applies directly to individual secondary buffers (the sound sources) and describes the way an individual source sounds within the surrounding environment.

### Setting the Listener Properties

The properties in the listener property set all work only on the primary buffer. Because of the way DirectSound works, you can only use this set through the property-set interface of a secondary buffer—not in the property-set interface of the primary buffer. You can use the property-set interface of *any* secondary buffer, but you must remember when using the listener properties that they affect the primary buffer as it's seen by all other secondary buffers. In other words, if you set a primary-buffer property in one secondary buffer interface, that primary property is set to the same value in all other secondary buffer interfaces.

The properties in the listener property set control the overall aural environment and affect the way all sound sources are perceived in the environment. You can, for example, set the quality of the overall reverberation and reflection as perceived by the listener.

You may want to think of listener properties as global properties that affect all sound sources, but that concept carries the implication that listener properties override sound-source properties. They don't: sound-source property settings are added to listener property settings. A better way to consider listener properties is as a set of baseline properties for all sound-source properties. Each sound-source property tweaks that baseline up or down for an individual sound source. And EAX's automatic distance management further scales sound-source properties depending on the source's position and directivity—unless you decide to disable it.

Because EAX adds reverberation and reflection independently to DirectSound's 3D effects, the sound sources retain their original 3D positional quality and have it enhanced by EAX's environmental audio effects.

To set primary-buffer properties, an application requests a new property-set interface for a secondary buffer—or it can use an existing property-set interface if it received one earlier while setting secondary-buffer properties. The application calls the `Set` member function on the interface and passes to it the listener property-set GUID, the property ID, and a pointer to the value it wants to set. EAX applies the new property setting to the primary buffer regardless of where the property was set.

## Setting the Sound-Source Properties

The properties in the sound-source property set allow the application to control how environmental effects are applied to each individual sound source. They control the amount of attenuation and tonal filtering applied to the source's direct and reflected sounds—which determines the amount of reverberation, obstruction, and occlusion the listener hears for the source.

An application can take as much or as little control over these properties as it cares to. If it doesn't set the properties at all, they're handled automatically by the EAX engine, which dynamically changes them as is appropriate for the source's directivity and its distance and orientation relative to the listener. An application can create additional effects by setting obstruction, occlusion, and reverberation properties for the sound source, and choose to either maintain or override EAX's automatic control. It can use higher-level effects such as occlusion or obstruction or bypass these higher-level effects by directly controlling the lower-level effects that are created by the higher-level controls: fundamental effects such as attenuation and tonal filtering.

To use the sound-source property set with a secondary buffer, an application requests a property-set interface for the buffer. When it gets the interface, it calls the `Set` member function on the interface and specifies the EAX sound-source GUID, specifies the property ID, and provides a pointer to the property value it wants to set. That value applies only to this secondary buffer and not to any other buffers.

# HOW EAX FITS INTO OPENAL

The first step in learning how to use EAX with OpenAL is to understand its place in the OpenAL environment. OpenAL has extensive documentation explaining the API, so we won't go into too many details here. We'll explore only the main concepts you'll need to understand how EAX fits in.

EAX comprises two different property sets. The first, called the *listener property set,* applies to OpenAL's Listener object and describes the listener's surrounding environment (the "room" in which the listener listens). The second set, called the *sound-source property set,* applies directly to OpenAL sources and describes the way an individual source sounds within the surrounding environment.

## SETTING THE LISTENER PROPERTIES

The properties in the listener property set control the overall aural environment and affect the way all sound sources are perceived in the environment. You can, for example, set the quality of the overall reverberation and reflection as perceived by the listener.

You may want to think of listener properties as global properties that affect all sound sources, but that concept carries the implication that listener properties override sound-source properties. They don't: sound-source property settings are added to listener property settings. A better way to consider listener properties is as a set of baseline properties for all sound-source properties. Each sound-source property tweaks that baseline up or down for an individual sound source. And EAX's automatic distance management further scales sound-source properties depending on the source's position and directivity—unless you decide to disable it.

Because EAX adds reverberation and reflection independently to OpenAL's 3D effects, the sound sources retain their original 3D positional quality and have it enhanced by EAX's environmental audio effects.

To set listener properties, an application queries for EAX support, retrieves a pointer to the EAXSet function, and then calls the EAXSet function while passing it the listener property-set ID, the property ID, and a pointer to the value it wants to set.

## SETTING THE SOUND-SOURCE PROPERTIES

The properties in the sound-source property set allow the application to control how environmental effects are applied to each individual sound source. They control the amount of attenuation and tonal filtering applied to the source's direct and reflected sounds—which determines the amount of reverberation, obstruction, and occlusion the listener hears for the source.

An application can take as much or as little control over these properties as it cares to. If it doesn't set the properties at all, they're handled automatically by the EAX engine, which dynamically changes them as is appropriate for the source's directivity and its distance and orientation relative to the listener. An application can create additional effects by setting obstruction, occlusion, and reverberation properties for the sound source, and choose to either maintain or override EAX's automatic control. It can use higher-level effects such as occlusion or obstruction or bypass these higher-level effects by directly controlling the lower-level effects that are created by the higher-level controls: fundamental effects such as attenuation and tonal filtering.

To use the sound-source property set, an application queries for EAX support, retrieves a pointer to the EAXSet function, and then calls EAXSet while specifying the EAX sound-source property ID, the ID of the source to be set, and a pointer to the property value it wants to set.

# USING EAX

EAX offers several different approaches to creating environmental effects. They range from high-level approaches where EAX does almost all of the work itself to low-level approaches where a programmer can directly tweak all the tiny details of environmental effects. In this section we'll look at the levels of control that EAX offers and at programming approaches to using those levels of control.

## LEVELS OF CONTROL

EAX provides an extensive set of environmental audio parameters that you can tweak through properties. These parameters are layered. Some are high level, providing large overall effects; others are low level, providing customized control over elementary aspects of the aural sensation. How much direct control of sound design you want determines what parameters you'll work with.

### High-Level Control

For excellent environmental audio effects with very little programming work, you can use high-level properties. The simplest property (found in the listener property set) lets you select an *environmental preset*—that is, to specify the type of room in which you want the listener and sound sources located. Once you specify the room using this single property, the EAX engine does all the rest of the work necessary to create the aural illusion of being in that room. It automatically controls the lower-level parameters to set the amount and quality of reverberation and reflection, to vary the reverberation/direct-sound ratio of each sound source as that source moves within the room, and much more.

All of the lower-level parameters are automatically computed for each source based on DirectSound's positional parameters and the environmental preset, which means that if you don't add obstruction or occlusion effects you don't even have to set sound-source properties. And because EAX uses a statistical reverberation model, you get convincing dynamic reverberation changes without having to provide a polygon-based description of room geometry.

If you'd like to tweak an environmental preset, you can use other high-level properties to change aspects of the room until you essentially create a new type of room. You can change the size of the room, the amount of air absorption, the directivity settings of each source, and other parameters. The low-level parameters affected by each of these high-level control methods are automatically adjusted by the EAX driver.

## Low-Level Control

If you have specific intentions about exactly how each effect should sound, you can gain complete control of elementary environmental audio parameters through EAX's low-level properties. These properties set the minutiae of environmental parameters, those that are normally controlled automatically by high-level properties. Low-level properties include listener properties that set reverberation decay time, reflection and reverberation delays, and reverberation diffusion. They also include sound-source properties that adjust levels and filters for each sound source.

Low-level properties directly control internal parameters that are additive to the parameters set by higher-level properties. The environmental preset sets the baseline parameters. Other high-level environment properties such as room size and air absorption tweak the environmental parameters one way or the other. And low-level properties tweak the environmental parameters even further in very localized ways.

## APPROACHES TO ENVIRONMENTAL AUDIO PROGRAMMING

Because EAX offers different levels of control, you can take different approaches to creating an audio environment in your program. These approaches each typically divide into two parts: sound design and programming. In sound design you decide what environmental audio effects you'd like to hear, where you'd like to hear them, and what quality those effects should have. Programming implements the elements of sound design and integrates them with the dynamic runtime events in the application. The sound designer can then adjust settings to get the acoustic environment sounding just right.

### Creating a Single Environment

In the easiest approach to environmental audio, you don't really need to spend any time with sound design other than deciding what kind of environment that you want your listener and sound sources to populate. To do so, you choose the EAX environmental preset that most closely matches the environment you want.

Once you're done with that, you really don't have much programming to do either. You simply specify the environmental preset you want using the listener property set, then let the EAX driver do all the rest of the work. This simple step by itself substantially enhances the listener's experience, adding life and reality to all game sounds (or the sounds of whatever application you're creating). It reinforces the listener's perception of sound-source distance and the depth of the sound scene as well.

If you'd like to go a step further, you can tweak the environmental preset properties using high-level listener properties to change the quality of the surrounding "room." You can also—independently of the environment parameter settings—add realism to individual sound sources by tuning the source's directivity parameters if required. The EAX engine automatically adjusts the reverberation intensity and tonal filtering for each source, taking into account its distance from the listener and its orientation and directivity.

## Creating a World of Multiple Environments

If your application moves the listener from environment to environment (such as going from room to room in a first-person 3D game), you can create multiple environments, each with its own unique set of acoustic properties. As the listener moves from one room to the next, the acoustic quality changes appropriately.

For sound design, you must first pick or design a predefined environmental preset for each room. If you'd like to modify individual presets to design a specific room, you can decide on settings for reverberation and for reverberation rolloff with distance and air absorption. This creates a custom environmental preset. If you think there will be overly abrupt acoustic transitions from one room to the next, you can design "transition environments," small environments located between rooms. A transition environment can have acoustic qualities that are halfway between its adjacent rooms so the acoustic changes are less abrupt when the listener comes through.

You can—just as in a single environment—tweak directivity parameters for individual sound sources if they require it. This requires no more work than it does in a single environment because it's a source parameter that applies to the source no matter what environment it resides in.

Programming your sound design requires identifying what environment your listener and sound sources are in. Because most game programs already have location mechanisms for their graphics, they can use these to define adjacent closed environments in the 3D world and to check on listener and source positions and test to see if they fall into one acoustic environment or another. If the program detects the listener moving from one environment to another, it sets the appropriate environmental preset.

Once a listener is in a room with sound sources, the program doesn't need to do any more work—the EAX engine performs all the necessary calculations for moving sources and listener. And if, after listening to the results, the sound designer wants to make changes, he can do so by changing just a few property values—changes don't require extensive reprogramming.

## Implementing Occlusion between Environments

In a world of multiple environments, there will be times when the listener is in one room and an audible sound source is in another room. In that case, the listener hears an occluded sound source transmitted through the wall between rooms. You can add occlusion effects to any sound source using EAX.

For sound design, you decide what kind of material makes up each partition between adjacent environments: how transmissive it is, how it attenuates high and low frequencies. You can choose or create a *material preset* (described in Appendix C) that matches each material. A material preset, like an environmental preset, controls a number of audio parameters. In this case, it controls low-level sound-source properties so they create the illusion of the sound coming through the specified material.

You can create a material preset from scratch by directly setting low-level sound-source properties, or you can choose a predefined material preset. The predefined presets specify materials such as concrete, wood, or a thin door. You can then tweak a predefined preset if you wish to get precisely the effect you want.

Programming occlusion requires detecting when a sound source is in an adjacent room to the listener, and then determining which partition separates source and listener. As in the last situation, a program can use its 3D graphics mechanisms to do much of this work. Whenever the program detects a sound source in an adjacent room to the listener and determines which partition separates them, the program applies the appropriate material preset to the sound source's occlusion properties. If the sound designer doesn't like the results, he can easily tune some material presets or, if necessary, directly adjust occlusion properties for each source.

## Implementing Obstruction within an Environment

When a sound source and the listener are in the same environment but an obstacle blocks the direct sound path from the source to the listener, the sound is obstructed. Its direct sound may be diffracted around the edges of the obstruction or transmitted through the obstruction if the object is acoustically transmissive. (Although both may occur, one of the two will usually overcome the other. Transmitted sound reaches the ear earlier than diffracted sound and the diffracted sound will then often blend into the reflections.) You can add obstruction effects to any sound source through EAX in much the same way you add occlusion effects—including using material presets when sound transmits through the obstacle.

The sound design for obstruction requires you to adjust obstruction settings to best simulate sound diffraction (how much attenuation and filtering is applied to sound waves as they bend around an obstacle) or—if the object has any acoustic transparency—the kind of material the object is made of. You determine the obstruction setting that best matches a diffraction effect or, if the object transmits sound, you create or choose a material preset for the object.

Programming requires positional detection that notices when an object appears between a sound source and a listener in the same room. When this occurs, the program sets the sound source's diffraction setting appropriately or—if the obstruction is acoustically transmissive—sets a material preset.

### Creating your own Low-Level Environmental Audio Effects

EAX's higher-level properties effectively create a full set of carefully designed and realistic environmental audio effects. There are times, however, when you may want something that goes beyond realistic for psychological impact—a disembodied spirit voice, for example, or sounds of battle that suddenly mute when the hero enters a mental trance. You can use EAX's low-level properties to create these kinds of effects to match your creative vision.

There are also times, if you're a committed sound designer with definite ideas about how effects should be implemented, when you may wish to implement, on your own, higher-level effects that EAX already provides. Although you may duplicate a lot of work that's already been done, EAX gives you the flexibility to do so through low-level properties. Because control at this level involves many personal and very technical issues, there's no recommended procedure for sound design and programming implementation.

## ACOUSTIC MODELS IN EAX

EAX properties tweak the parameters of underlying acoustic models that the EAX engine uses to create environmental audio. These models determine the quality of the acoustic environments you create and the way sound sources interact with the environments. To use EAX's low-level properties well, it's important to understand the components of the models and how they work together. This section describes acoustic model components and how EAX properties control them.

### ENVIRONMENT REVERBERATION MODEL

In EAX, an environment is described by parameter settings that define the acoustical quality of the reverberation. It's also described by a number of properties whose settings apply to all sound sources in the environment.

## Reverberation Response Model

The reverberation response (shown in Figure 1) is divided into three temporal sections respectively labeled **Direct** (direct path), **Reflections** (initial reflections) and **Reverb** (exponentially decaying late reverberation). The temporal division between these components is defined by the parameters **Reflections Delay** and **Reverb Delay.** The **Decay Time** is derived from the slope of the late reverberation (Reverb) decay, and defined as the time that Reverb takes to decay by 60 decibels (dB).

This describes the response at the listener's position for a particular position of the source, assuming that the two are in the same environment (the "current" environment). All reflection and reverberation parameters therefore refer to the listener's room—not to the source's room if it is different (a case of occlusion).

The reverberation response is defined by the following parameters:

- The energy in each of the three sections: Direct, Reflections, and Reverb at low frequencies
- The Reflections Delay and the Reverb Delay
- The "Direct filter," a low-pass filter that affects the Direct component by reducing its energy at high frequencies
- The "Room filter," a low-pass filter that affects the Reflections and Reverb components identically by reducing their energy at high frequencies
- The Decay Time at low and high frequencies
- The Diffusion and the Size of the room (defined below)

The definition of “low” and “high” frequencies is given below in the section “Spectral Effects”.

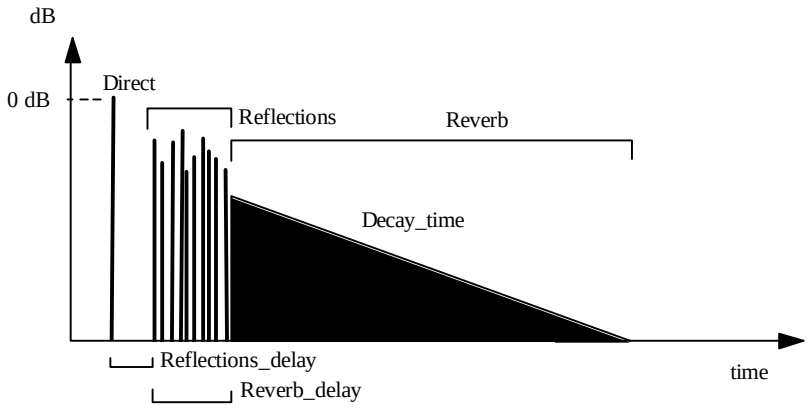


*Figure 1: Reverberation response model*

## Distance Effects

Both OpenAL and DirectSound have the notions of **Minimum Distance**, **Maximum Distance**, and **Rolloff Factor**. It applies an attenuation to the source signal according to source-listener distance as follows:

- If distance ≤ Minimum_Distance, the attenuation is 0 dB (no attenuation).
- If Minimum_Distance ≤ distance ≤ Maximum_Distance, the attenuation expressed in dB is -6 dB for each doubling of distance if the Rolloff Factor is set to 1.0. For values different from 1.0, the Rolloff Factor acts as a multiplier applied to the source-listener distance (diminished by the minimum distance).
- If distance ≥ Maximum_Distance, the attenuation no longer varies with distance.

In natural environments, the total intensity of the reflected sound received by the listener varies little vs. source-listener distance, and therefore the direct-to-reflected energy ratio decreases with distance. This provides an essential cue to the listener for assessing the distance of the source.

In general the reflected intensity actually decays somewhat with increasing distance, although not as fast as the direct path intensity. EAX provides two methods for automatically attenuating the reflected sound (Reflections and Reverb) according to source-listener distance:

- The first (and default) method simulates the natural rolloff of reverberation vs. distance in typical rooms. This method is based on a physically proven statistical reverberation model that allows predicting the intensity and tonal characteristics of reflections and reverberation in natural environments according to source-listener distance, room size, reverberation decay time and source directivity.
- The second method for controlling the rolloff of reflected sound vs. distance uses a parameter called **Room Rolloff Factor** that affects the reflected sound component instead of the direct-path sound component. The default value of Room Rolloff Factor is 0.0, which implies that the reflected intensity does not vary with distance, while the default value for the (Direct) Rolloff Factor is 1.0. Setting Room Rolloff Factor to 0.5 instead would imply that the reflected sound is also attenuated with increasing distance, although not as fast as the direct path.

The Minimum Distance and Maximum Distance act on the reflected sound as they do on the direct component. EAX 2.0 also provides an additional parameter to control the attenuation at high frequencies caused by the propagation medium. This parameter is called **Air Absorption HF** and is expressed in dB per meter. You can, for example, increase Air Absorption HF in order to simulate higher humidity in the air.

## Environment Presets

The Environment's reverberation quality is defined in a manner that applies to all sound sources in the Environment:

> **An Environment is characterized by the values of all reverberation response parameters when distance = Minimum_Distance. This defines an "Environment preset."**

The only reverberation response parameters that can vary automatically with distance in EAX are the intensities of the three temporal sections: Direct, Reflections and Reverb. At Minimum Distance, the intensity level of the Direct component is 0 dB (no attenuation with respect to the source signal, except for the attenuation due to DirectSound's Volume setting for that source, which affects Direct, Reflections and Reverb identically). Note that the intensity level of Reverb or Reflections is defined relative to the *source signal* (not relative to the Direct-Path component attenuated by DirectSound's rolloff effect).

EAX provides a list of predefined environment presets in order to make work easier for the application developer or content author.

## Environment Diffusion and Size

Diffusion and Room Size are familiar parameters of artificial reverberation processors. In EAX, **Environment Diffusion** controls the echo density in the Reverb section of the response (not the Reflections). **Environment Size** is expressed in meters and affects (by default) the delay and intensity of reflections and reverberation, as well as the reverberation decay time. When Room Size is set to about 2 meters or less, the reverberation also takes on a spectral coloration characteristic of small rooms.

The Room Size parameter in EAX works as a high-level parameter that allows tuning an environment preset by adjusting several lower-level parameters simultaneously in order to reproduce the physical effect of increasing the dimensions and cubic volume of a room. EAX also allows programmers to customize the effect of Room Size by selecting which low-level parameters are affected.

## Spectral Effects

All spectral effects in EAX are controlled by specifying an attenuation at a reference high frequency of 5 kHz. All low-pass effects are specified as high-frequency attenuations in dB relative to low frequencies. This manner of controlling low-pass effects is similar to using a graphic equalizer (controlling levels in fixed frequency bands). It allows the sound designer to predict the overall effect of combined (cascaded) low-pass filtering effects by adding together the resulting attenuations at 5 kHz. This method of specifying low-pass filters is also used in the definition of the EAX Occlusion and Obstruction properties and in the source directivity model as described in the next section.

## Reverberation Decay

Reverberation decay is the result of acoustic energy absorption by a room's surfaces and the propagation medium. Each time a sound wave bounces off a surface, it decreases in amplitude until there is negligible reflected sound. If room surfaces are acoustically "live," they absorb very little acoustic energy, reflected sounds diminish little each time they bounce, and the reverberation (the combination of all reflected sound) takes a long time to decay. If room surfaces are acoustically "dead," reverberation decays very quickly as acoustic energy is absorbed more quickly.

For a room of a given size, the Decay Time property effectively sets the average acoustic properties of the virtual room's surfaces by setting the time in the number of seconds it takes for the reverberation to diminish by 60 dB. For a room having certain given wall materials, the Decay Time is roughly proportional to the dimension of the room expressed in meters.

The acoustic reflectivity of a surface material is not always even across all frequencies. Many surfaces reflect more low and middle frequencies while absorbing high frequencies. The high-frequency decay time in EAX is defined as the decay time at the reference high frequency of 5 kHz. In practice, it is controlled via a multiplicative factor applied to the low-frequency Decay Time (called **Decay HF Ratio**). As a result, increasing the (low-frequency) Decay Time implies a proportional increase of the high-frequency decay time. In order to avoid unnaturally long decay times at high frequencies, EAX 2.0 offers the option of limiting the high-frequency decay time to a value controlled by the setting of the **Air Absorption HF** parameter (in large rooms, it is the air absorption that determines the high-frequency decay time).

## INTERACTION BETWEEN SOUND SOURCES AND ENVIRONMENT

Individual sound sources within the acoustic environment each have their own aural behavior that's determined by acoustic models.

### Source Processing Model

EAX's processing model for each sound source comprises an attenuation and a low-pass filter that are applied independently to the direct path and the reflected sound (see Figure 2). All the sound-source properties in EAX have the effect of adjusting these attenuation and filter parameters relative to the Environment settings.

The Environment settings are described by the listener properties that we just discussed. The settings apply for an omnidirectional sound source located at Minimum Distance, in the absence of Occlusion or Obstruction. (Note that DirectSound allows for the setting the Minimum Distance and the Maximum Distance differently for each sound source).

The EAX 2.0 sound-source property set defines a variation relative to the baseline setting described by the listener properties. This variation results from several combined effects:

- Distance-dependent effects (statistical reverberation model, Room_Rolloff factor, and Air Absorption) as already described above
- Enhanced source directivity model (frequency-dependent source directivity)
- Occlusion and obstruction effects
- Manual adjustment of the Direct and Room attenuations and filters.

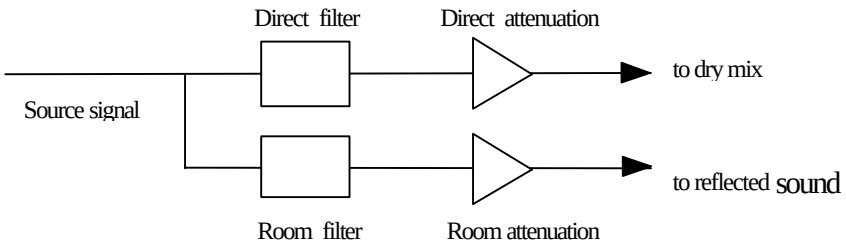


*Figure 2: Source processing model in EAX 2.0 (not showing 3D positional functions)*

## Orientation and Directivity

DirectSound and OpenAL both represent directive sound sources using the concept of *Sound Cones*. A source radiates its maximum intensity within the *Inside Cone* (in front of the source) and its minimum intensity in the *Outside Cone* (in back of the source). A sound source can be made more directive by making the Outside Cone Angle wider or by reducing the Outside Cone Volume.

EAX enhances this model by allowing programmers to further reduce the Outside Volume at high frequencies in order to make the sound source more directive at high frequencies than at low frequencies. As a result, when the listener is located away from the main radiation axis of the source, the sound is not only attenuated but also filtered. This feature can be used to obtain a more natural simulation of directive sources (such as voices or horns, for example).

The orientation of the source with respect to the listener modifies the intensity and color of the direct component, but it does not affect the reflected sound. A more directive source, however, radiates less total sound power into the room. EAX reproduces this natural property of room reverberation automatically by default. If the source is made more directive at high frequencies, the reverberation is filtered accordingly as it is in natural environments. This can also be an important factor in improving the naturalness of the sound scene.

## Occlusion and Obstruction

The concepts of Occlusion and Obstruction enable applications to reproduce the effects of obstacles standing between a source and the listener. Sounds can be heard through walls from other rooms, from around corners, through open and closed doors and windows, and from behind other objects. These sounds are different from the same sounds unobstructed, and this difference helps give a listener detailed information about the environment in which he or she is immersed and the sound-emitting objects within it. The Occlusion and Obstruction properties allow sources to be made to sound as if they are in other rooms, behind doors, around corners, or in any other way muffled by an obstacle.

Sounds that are transmitted through material structures undergo a frequency dependent attenuation. This attenuation usually has a low-pass character with the amount of attenuation and the slope of the high frequency rolloff being dependent on the material, thickness, and construction of the partition.

Sounds that travel around corners and through openings also undergo a low-pass attenuation due to a phenomenon known as diffraction. The effect of an obstacle will depend on the size of that obstacle with respect to the wavelength of the sound. If small compared to the wavelength, it will have very little effect. Thus low frequencies tend to travel around corners or swallow obstacles, and are not as attenuated by obstructions as high frequencies are. As a rule of thumb, the more acute the angle a sound path must make to travel around an obstacle, the greater the amount of high-frequency attenuation.

EAX distinguishes two cases:

- **Obstruction** of sound by an object between the listener and the source, all contained within a common environment. The Direct path can reach the listener via *diffraction around the obstacle* and/or via *transmission through the obstacle*. In both cases, the direct path is muffled (low-pass filtered) but the reflected sound from that source is unaffected (because the source radiates in the listener's environment and the reverberation is not blocked by the obstacle). Most often the transmitted sound is negligible and the low-pass effect only depends on the position of the source and listener relative to the obstacle, not on the transmission coefficient of the material. In the case of a highly transmissive obstacle (such as a curtain), however, the sound that goes through the obstacle may not be negligible compared to the sound that goes around it.

- **Occlusion** of sound by a partition or wall separating two environments (rooms). Sounds that are in a different room or environment can reach the listener's environment by transmission through walls or by travelling through any openings between the sound source's and the listener's environments. Before these sounds reach the listener's environment they have been affected by the transmission or diffraction effects, therefore both the direct sound and the contribution by the sound to the reflected sound in the listener's environment are muffled. In addition to this, the element that actually radiates sound in the listener's environment is not the original sound source but the wall or the aperture through which the sound is transmitted. As a result, the reverberation is usually more attenuated by occlusion than the direct component because the actual radiating element is more directive than the original source.

The perceptual effects of the two situations, Obstruction and Occlusion, are similar in nature: the sound undergoes an amount of low-pass attenuation. The difference between the two situations is that the reflected sound remains unaffected in the case of Obstruction, while it is affected in the case of Occlusion. In order to enable this functionality, it would in principle have been sufficient to expose the Direct and Room attenuations/filters defined in the previous section. However, programming these effects in an interactive application would have been inconvenient because it would systematically require updating four parameters simultaneously (for occlusion) or at least two parameters (for obstruction).

For this reason, EAX 2.0 provides a more hierarchized interface to control the effects of obstacles:

- It distinguishes Obstruction (where only the direct path is muffled) and Occlusion (where both the direct path and the reflected sound are muffled).
- Both effects are controlled by specifying the high-frequency attenuation (the main perceived effect) at the reference frequency of 5 kHz, optionally accompanied by an attenuation at low frequencies. In order to allow controlling both effects via a single knob, the second is relative to the first as illustrated in Figure 3. In the case of Occlusion, these two effects are also accompanied by an adjustable attenuation of the reflected sound compared to the direct component.

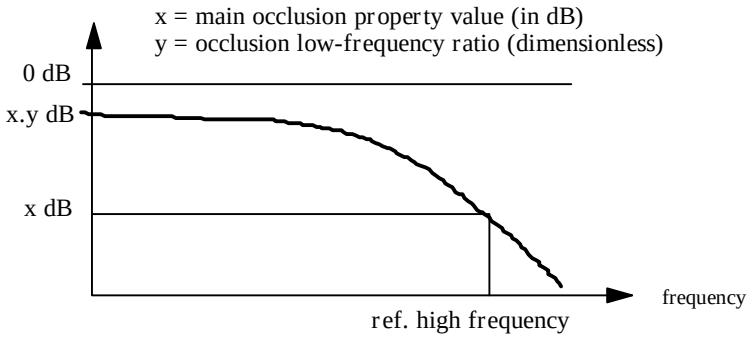


*Figure 3: Occlusion or Obstruction filter control model. The reference high frequency is set to 5 kHz.*

## Using the Occlusion and Obstruction properties

The Occlusion property, which you use when a sound source is in a different environment than the listener, might be used in the following example scenario. A loud moving sound source is in a room that has a closed door leading to a hallway. If the listener is in the hallway outside the room the sound source could have its Occlusion property set to –40 dB, which makes it sound as if it is behind a thick wall. When the source passes in front of the doorway to the room, the Occlusion property value can be changed to –30 dB, which will make the sound slightly more clear since the door is thinner than the wall. While the door opens the value can be gradually changed to –8 dB, and if the source enters the room the value can again be changed to 0 dB so the sound will be completely clear. Although this could all be done by just manipulating the main Occlusion property, the simulation can also be refined by assigning a particular material preset to the wall (which sets the secondary Occlusion properties to values different from their default value).

The Obstruction property might be used in the following way. If the listener and the sound source are in the same room and there is nothing between them, then the Obstruction property for the sound source can be set to 0 dB (its default value). If the sound source moves behind a large object (a large pillar for example), the Obstruction property could be set to a value of –25 dB. As the sound source moves further behind the obstacle, the Obstruction property value can be progressively lowered. The value of the Obstruction property can be related to the angle at which the sound must travel around to reach the listener (the more acute the angle, the more the sound is muffled). If the source, while behind the obstacle, moves into another room, Occlusion can be used simultaneously with Obstruction.

# SAMPLE CODE: CREATING BUFFERS AND INTERFACES IN DIRECTSOUND

To use EAX in an application, you must first start DirectSound. You then create appropriate sound buffers and create three interfaces for each buffer:

- A standard interface to control fundamental buffer behavior (playback volume, frequency, and so on)
- A 3D interface to control 3D behavior (position, velocity, and so on)
- A property-set interface to set EAX properties for the buffer

To do so, follow the steps described below, illustrated with sample code.

## CREATING THE STANDARD DIRECTSOUND INTERFACES

First you need to create the standard 2D and 3D interfaces for the primary buffer and the secondary buffers.

### Primary Buffer

1. Instantiate a DirectSound object to start DirectSound, then use the IDirectSound interface to control the DirectSound object:

```
LPDIRECTSOUND pDirectSoundObj;
    DirectSoundCreate(NULL, &pDirectSoundObj, NULL);

    pDirectSoundObj->SetCooperativeLevel(hWnd, DSSCL_EXCLUSIVE);
```

Creating the DirectSound object gives the application a pointer to the interface for that object. You can use the interface as shown in line 3 of the sample above to set DirectSound's status: its cooperative level, number of speakers, and so forth.

2. Instantiate a new 3D primary buffer :

```
LPDIRECTSOUNDBUFFER pPrimaryBuf;
    DSBUFFERDESC desc;
    ...
    desc.dwFlags = DSBCAPS_PRIMARYBUFFER | DSBCAPS_CTRL3D;
    pDirectSoundObj->CreateSoundBuffer(&desc, &pPrimaryBuf,
        NULL);
```

The flags DSBCAPS_PRIMARYBUFFER and DSBCAPS_CTRL3D specify a 3D primary buffer.

Creating the primary buffer gives the application a pointer to the buffer's standard interface. Note that the new buffer you create replaces the default primary buffer created when you instantiated the DirectSound object.

3. Get a 3D interface (called IDirectSound3DListener) for the new primary buffer:

```
LPDIRECTSOUND3DLISTENER pListener;
    pPrimaryBuf->QueryInterface(IID_IDirectSound3DListener
        (void**)&pListener);
```

You can use this interface to control listener-specific 3D aspects of the primary buffer behavior—listener orientation, position, velocity, and so forth.

## Secondary Buffers

1. Instantiate a secondary buffer for each sound source in the 3D aural world:

```
LPDIRECTSOUNDBUFFER pSecondaryBuf[n];
    DSBUFFERDESC desc;
    ...
    desc.dwFlags = DSBCAPS_CTRL3D | ...
    pDirectSoundObj->CreateSoundBuffer(&desc, &pSecondaryBuf[i],
        NULL);
```

   The flag DSBCAPS_CTRL3D makes the buffer a 3D buffer.

   Each secondary buffer creation gives the application a pointer to the buffer's standard interface.

4. Get a 3D-secondary-buffer interface (called IDirectSound3DBuffer) for each new secondary buffer:

```
LPDIRECTSOUND3DBUFFER p3Dbuf[n];
    ...
    pSecondaryBuf[i]->QueryInterface(IID_IDirectSound3DBuffer,
        (void **)&p3Dbuf[i]);
```

Each of these interfaces controls 3D aspects of a secondary buffer's behavior.

## CREATING THE PROPERTY-SET INTERFACES

Once you have obtained the standard interfaces, you can establish the EAX listener property set interface (on the primary buffer) and source property set interfaces (on the secondary buffers).

### Primary Buffer

1. Establish a property-set interface for the primary buffer (the listener). Because a DirectSound quirk doesn't allow an application to use a property set directly with the primary buffer interface, your application must use one of the secondary buffer's property-set interfaces to set EAX listener properties. You can get a pointer to an interface from DirectSound:

```
LPKSPROPERTYSET pEAXListener;
    p3Dbuf[0]->QueryInterface(IID_IKsPropertySet,
        (void**)&pReverb);
```

   You get a property-set interface for a buffer by calling `QueryInterface` on the buffer's 3D interface. It writes a pointer to the property-set interface. (This example uses the first secondary buffer you created, buffer 0. You could, in fact, use any secondary buffer.)

You can use the property-set interface to set EAX listener properties by specifying the EAX GUID and appropriate listener property numbers as described later.

Note that if you ask for a property-set interface and the audio driver doesn't support property sets, `QueryInterface` will fail.

## Secondary Buffers

1. Get a property-set interface for the secondary buffer of each sound source you want to control:

```
LPKSPROPERTYSET pEAXSource[n];
    ...
    p3Dbuf[i]->QueryInterface(IID_IKsPropertySet, (void**)
        &pReverbBuffer[i]);
```

   This example gets a property-set interface for each secondary buffer.

You can use each secondary buffer's property-set interface to set EAX properties by specifying the EAX GUID and a property number as described in the next section.

Note that you can skip this step completely if you don't need to control occlusion or obstruction effects, customize distance or directivity effects, or adjust the reverberation amount on individual sources. Your program then leaves control to EAX which, by default, automatically adjusts attenuations and filters for each sound source according to its distance from the listener, its directivity, and the current setting of the listener properties.

# SAMPLE CODE: SETTING EAX PROPERTIES IN DIRECTSOUND

Once you've created primary and secondary buffers and established three interfaces (standard, 3D, and property-set) for each buffer you want to control, you can set EAX properties through DirectSound's property-set interface.

A property-set interface offers three methods: `QuerySupport`, `Set`, and `Get`. These methods all require a GUID that identifies a property set and the ID number of a property in that set. When you use these methods with EAX, it's important to remember that EAX is really two property sets:

- **The EAX sound-source property set**, which applies to a sound source (a secondary buffer). Its GUID is the constant `DSPROPSETID_EAX_ListenerProperties`, defined in the header file `eax.h`.
- The **EAX listener property set,** which applies to the listener (the primary buffer). Its GUID is the constant `DSPROPSETID_EAX_BufferProperties`, defined in the header file `eax.h`.

## QUERYING FOR EAX SUPPORT

Before your application tries to work with EAX properties, you may want it to check to be sure that the audio driver supports the two EAX property sets. To do so, you call `QuerySupport` on the property-set interface of any buffer.

`QuerySupport` can query to see if either EAX property set exists or it can query to see if an individual EAX property is supported. In both cases, `QuerySupport` requires the application to pass it the GUID of the property set being queried along with a property number. If you want to query for the listener property set, pass the listener property set GUID along with the constant `DSPROPERTY_EAXLISTENER_ALL` as the property number. If you want to query for the sound-source property set, pass the sound-source property set GUID along with the constant `DSPROPERTY_EAXBUFFER_ALL` as the property number.

The following sample code queries the EAX interface pointed to by pReverb. It asks if it supports the EAX listener property set, then checks to see if the value returned includes flags that show support for getting and setting all properties within the set:

```
ULONG support=0;
if (FAILED(pReverb->QuerySupport(DSPROPSETID_EAX_ListenerProperties,
    DSPROPERTY_EAXLISTENER_ALL, &support)))
        AfxMessageBox("EAX 2.0 not supported");
if ((support & (KSPROPERTY_SUPPORT_GET|KSPROPERTY_SUPPORT_SET)) !=
    (KSPROPERTY_SUPPORT_GET|KSPROPERTY_SUPPORT_SET))
        AfxMessageBox("EAX 2.0 not supported");
```

Note that you can skip querying for EAX support and try setting or getting a property directly. If the audio driver doesn't support the EAX property sets, DirectSound will report an error that you can handle as you see fit.

## SETTING AND TWEAKING A LISTENER ENVIRONMENT

The highest-level property in the EAX listener property set is Environment. This property defines the size and acoustic quality of the apparent room in which the listener is located. When you set Environment, EAX automatically sets values for all the other listener properties, which are defined internally as part of the environment but can be further modified.

To set Environment, use the property-set interface of any secondary buffer. (The set affects the primary buffer's operation, not the secondary buffer's operation.) You call Set on the interface:

```
DWORD envId=EAX_ENVIRONMENT_AUDITORIUM;
pReverb->Set(DSPROPSETID_EAX_ListenerProperties,
    DSPROPERTY_EAXLISTENER_ENVIRONMENT,NULL,0,&envId,sizeof(DWORD));
```

The first argument you pass is the GUID of the EAX listener property set. The second argument is a constant (defined in eax.h) that identifies the property. The third and fourth arguments define a parameter, which isn't used by EAX, so they receive NULL and 0. The fifth and sixth arguments pass the property's new value. The fifth is a pointer to the value, the sixth gives the size of the value.

Once you set an environment, you can tweak the environment by setting any of the other listener properties. Set them just as you do Environment. This example sets Reverb:

```
LONG Reverb=-600;
    pReverb->Set(DSPROPSETID_EAX_ListenerProperties,
        DSPROPERTY_EAXLISTENER_REVERB,NULL,0,&Reverb,sizeof(LONG));
```

## SETTING ALL THE LISTENER PROPERTIES AT ONCE

To set all the listener properties at once, you call `Set` on a property-set interface just as you did in the last example, but the ID you use to specify a property—DSPROPERTY_EAXLISTENER_ALL—specifies *all* listener properties in this case. And the value you pass to the `Set` method is an array of values for all listener properties instead of a single value:

```
EAXLISTENERPROPERTIES props={-789,-
476,0.0f,4.32f,0.59f,0,0.020f,8,0.030f,
    EAX_ENVIRONMENT_AUDITORIUM,21.6f,1.00f,-5.00f,0x0000002F};
pReverb->Set(DSPROPSETID_EAX_LiastenerProperties,
    DSPROPERTY_EAXLISTENER_ALL,
    NULL,0,&props,sizeof(EAXLISTENERPROPERTIES));
```

The values in the array apply to the listener properties in the order defined by the type definition of EAXLISTENERPROPERTIES in the header file `eax.h`.

## SETTING A SOUND-SOURCE PROPERTY

To set a sound-source property, use the property-set interface of the buffer you want to affect. You then call `Set` on the interface, as in the following example:

```
LONG Room=0;
pReverbBuffer[i]->Set(DSPROPSETID_EAX_BufferProperties,
    DSPROPERTY_EAXBUFFER_ROOM, NULL, 0, &Room, sizeof(LONG));
```

The first argument is the GUID of the EAX sound-source property set; the second is the ID of the property you want to set (a constant defined in `eax.h`), in this case Environment. The remaining arguments are set to ignore parameter and point to a property value, just as in the previous examples.

## SETTING ALL THE SOURCE PROPERTIES AT ONCE

To set all the source properties at once, you call `Set` on a property-set interface just as you did in the last example, but the ID you use to specify a property—DSPROPERTY_EAXBUFFER_ALL—specifies *all* source properties in this case. And the value you pass to the `Set` method is an array of values instead of a single value:

```
EAXBUFFERPROPERTIES props={0,0,0,-600,0.0f,0,0.0f,0,0.25f,0.5f,
    -1200,1.0f, 0x00000007};
pReverb->Set(DSPROPSETID_EAX_BufferProperties,
    DSPROPERTY_EAXBUFFER_ALL, NULL, 0, &props,
sizeof(EAXBUFFERPROPERTIES));
```

The values in the array apply to the source properties in the order defined by the type definition of EAXBUFFERPROPERTIES in the header file `eax.h`.

## GETTING A PROPERTY VALUE

You can, at any time, check to see the current value of any EAX property. To do so, call `Get` on a primary or secondary buffer's property-set interface and identify the property you want by GUID and property ID. For example:

```
LONG Reverb;
ULONG revsize;
pReverb->Get(DSPROPSETID_EAX_ListenerProperties,
    DSPROPERTY_EAXLISTENER_REVERB,NULL,0,&reverb,sizeof(LONG),
        &revsize);
```

This example gets the value of the Reverb property in the listener property set. `Get` returns the value of Reverb in the variable `Reverb` and writes the number of bytes it wrote into `Reverb` into the variable `revsize`.

# SAMPLE CODE: SETTING EAX PROPERTIES IN OPENAL

Once you've initialized OpenAL, you are ready to set listener or source properties in OpenAL. There are only two functions required to use EAX once it has been detected – EAXGet and EAXSet. The application selects between source and listener properties by passing the appropriate ID code to the Get or Set function – either PROPSETID_EAX_ListenerProperties or PROPSETID_EAX_SourceProperties.

## QUERYING FOR EAX SUPPORT

Before your application tries to work with EAX properties, you will have to check to make sure EAX is supported, and establish pointers to the EAXGet and EAXSet functions. This is easy to do using the OpenAL functions "alIsExtensionPresent" and "alGetProcAddress." The following sample code queries for EAX support and sets the function pointers if EAX is available:

```
PropSetFunction pfPropSet = NULL;
PropGetFunction pfPropGet = NULL;

if (alIsExtensionPresent((ALubyte *)"EAX") == AL_TRUE)
{
    pfPropSet = alGetProcAddress((ALubyte *)"EAXSet");
    pfPropGet = alGetProcAddress((Alubyte *)"EAXGet");
}
```

## SETTING AND TWEAKING A LISTENER ENVIRONMENT

The highest-level property in the EAX listener property set is Environment. This property defines the size and accoustic quality of the apparent room in which the listener is located. When you set Environment, EAX automatically sets values for all the other listener properties, which are defined internally as part of the environment but can be further modified.

To set Environment, use the EAXSet function as follows:

```
unsigned long ulEAXVal;


if (pfPropSet != NULL)
{
ulEAXVal = EAX_ENVIRONMENT_GENERIC;
    pfPropSet(PROPSETID_EAX_ListenerProperties,
DSPROPERTY_EAXLISTENER_ENVIRONMENT, 0, &ulEAXVal, sizeof(unsigned
long));
}
```

The first argument is the ID of the listener property set. The second argument is a constant (defined in eax.h) which identifies the property to be altered. The third argument is not used for listener properties. The fourth argument is a pointer to the data to be get/set. The fifth argument is the size of the data.

## SETTING ALL THE LISTENER PROPERTIES AT ONCE

To set all the listener properties at once, you call set using the listener property-set ID as in the last example, but the ID you use to specify a property – DSPROPERTY_EAXLISTENER_ALL – specifies all listener properties in this case. The value you pass to the Set method is an array of values for all listener properties instead of a single value:

```
EAXLISTENERPROPERTIES EAXVal = { -789, -476, 0.0f, 0.59f, 0, 0.020f, 8,
0.030f, EAX_ENVIRONMENT_AUDITORIUM, 21.6f, 1.00f, -5.00f, 0x0000002F };


if (pfPropSet != NULL)
{
    pfPropSet(PROPSETID_EAX_ListenerProperties,
DSPROPERTY_EAXLISTENER_ALL, 0, &EAXVal, sizeof(EAXLISTENERPROPERTIES));
}
```

## SETTING A SOURCE PROPERTY

The procedure for setting a sound-source property is the same as for setting a listener property, except that the source property ID is passed-in as the first parameter, and the OpenAL source ID is passed-in as the third parameter. Here is an example where the source "room" parameter is being set:

```
long lEAXVal;


if (pfPropSet != NULL)
{
lEAXVal = 0;
    pfPropSet(PROPSETID_EAX_SourceProperties,
DSPROPERTY_EAXBUFFER_ROOM, 0, &lEAXVal, sizeof(long));
}
```

## SETTING ALL THE SOURCE PROPERTIES AT ONCE

To set all the source properties at once, the property ID to use is DSPROPERTY_EAXBUFFER_ALL. Here is an example:

```
EAXBUFFERPROPERTIES EAXVal = { 0, 0, 0, -600, 0.0f, 0, 0.0f, 0, 0.25f,
0.5f, -1200, 1.0f, 0x00000007 };


if (pfPropSet != NULL)
{
    pfPropSet(PROPSETID_EAX_SourceProperties, DSPROPERTY_EAXBUFFER_ALL,
0, &EAXVal, sizeof(EAXBUFFERPROPERTIES));
}
```

## GETTING A PROPERTY VALUE

You can, at any time, check to see the current value of any EAX property. To do so, call Get using the source or listener proptery-set ID and the property ID you want to retrieve. For example:

```
unsigned long ulEAXVal;


if (pfPropGet != NULL)
{
    pfPropGet(PROPSETID_EAX_ListenerProperties,
DSPROPERTY_EAXLISTENER_ENVIRONMENT, 0, &ulEAXVal, sizeof(unsigned
long));
}
```

This example retrieves the listener environment setting.

# AN OVERVIEW OF EAX PROPERTIES

Before looking at detailed descriptions of the EAX properties, you can see an overview of both the listener and sound-source property sets in this short section. You'll find details for the properties in the following section.

## PROPERTY CONVENTIONS

All intensity levels are defined relative to the intensity level of the source signal. It's important to realize that the source signal is *not* the same thing as the direct-path sound from the source. The direct-path sound may be attenuated according to distance or directivity, and so may be less intense than the source. The intensity of the sound source is measured at 0 dB. All other intensities are relative to this point.

All times are expressed in seconds. All intensity levels or relative attenuations are expressed in hundredth of decibels (millibels, mB). The relation between the gain expressed in millibels and the linear amplitude gain is:

gain_mB = 2000 * log10(gain)

Some properties appear with the same name in both the sound-source and listener property sets. Whenever this happens, the two values simply combine additively—in other words the listener property acts as an offset that applies to all sources.

When using DirectSound, some properties only operate if Direct Sound's 3D mode is enabled. This is indicated for each property in the following tables by the characters *3D* in the left column.

## TABLE OF LISTENER PROPERTIES

| | PROPERTY NAME | TYPE | RANGE | DEFAULT |
|---|---|---|---|---|
| | All | array | | |
| | Environment | DWORD | [0, 25] | 0 |
| | Environment Size | FLOAT | [1.0, 100.0] | (*) meters |
| | Environment Diffusion | FLOAT | [0.0, 1.0] | (*) |
| | Room | LONG | [–10000, 0] | (*) mB |
| | Room HF | LONG | [–10000, 0 ] | (*) |
| | Decay Time | FLOAT | [0.1, 20.0] | (*) seconds |
| | Decay HF Ratio | FLOAT | [0.1, 2.0] | (*) |
| | Reflections | LONG | [–10000, 1000] | (*) mB |
| | Reflections Delay | FLOAT | [0.0, 0.3] | (*) seconds |
| | Reverb | LONG | [–10000, 2000] | (*) mB |
| | Reverb Delay | FLOAT | [0.0, 0.1] | (*) seconds |
| *3D* | Room Rolloff Factor | FLOAT | [0.0, 10.0] | 0.0 |
| | Air Absorption HF | FLOAT | [–100.0, 0.0] | –5.0 mB/m |
| | Flags | DWORD | [0x0, 0x2F] | (*) |
| | • Decay Time Scale | Flag bit | | TRUE |
| | • Reflections Scale | Flag bit | | TRUE |
| | • Reflections Delay Scale | Flag bit | | TRUE |
| | • Reverb SCALE | Flag bit | | TRUE |
| | • Reverb Delay Scale | Flag bit | | TRUE |
| | • Decay HF Limit | Flag bit | | (*) |

(*) means that the default value depends on Environment.

## TABLE OF SOUND-SOURCE PROPERTIES

| | PROPERTY NAME | TYPE | RANGE | DEFAULT |
|---|---|---|---|---|
| | All | array | | |
| | Direct | LONG | [–10000, 1000] | 0 mB |
| | Direct_HF | LONG | [–10000, 0] | 0 mB |
| | Room | LONG | [–10000, 1000] | 0 mB |
| | Room_HF | LONG | [–10000, 0] | 0 mB |
| *3D* | Obstruction | LONG | [–10000, 0] | 0 mB |
| *3D* | Obstruction_LF_ratio | FLOAT | [0.0, 1.0] | 0.0 |
| *3D* | Occlusion | LONG | [–10000, 0] | 0 mB |
| *3D* | Occlusion_LF_ratio | FLOAT | [0.0, 1.0] | 0.25 |
| *3D* | Occlusion_Room_ratio | FLOAT | [0.0, 10.0] | 0.5 |
| *3D* | Room_rolloff_factor | FLOAT | [0.0, 10.0] | 0.0 |
| *3D* | Air_absorption_factor | FLOAT | [0.0, 10.0] | 1.0 |
| *3D* | Outside_volume_HF | LONG | [–10000, 0] | 0 mB |
| *3D* | Flags | DWORD | [0x0, 0x7] | 0x7 |
| | • Direct HF Auto | Flag bit | | TRUE |
| | • Room Auto | Flag bit | | TRUE |
| | • Room HF Auto | Flag bit | | TRUE |

# DETAILED EAX PROPERTY DESCRIPTIONS

EAX contains two property sets: the listener property set and the sound-source property set. This section describes, in detail, the properties of each. To understand how lower-level properties described here combine to create the effects controlled by higher-level properties, you may want to read the previous section "Acoustic Models in EAX."

You'll find all basic structures, types, and constants necessary for implementing EAX in the header file `eax.h`. You'll also find the minimum, maximum, and default values for each property defined here.

## THE LISTENER PROPERTY SET

The listener property set contains fourteen properties that apply through a secondary-buffer property-set interface to the primary buffer. To use these properties, you must specify the property-set GUID `DSPROPSETID_EAX_ReverbProperties`.

### Environment

| | |
|---|---|
| Specify using this ID | `DSPROPERTY_EAXLISTENER_ENVIRONMENT` |
| Value type | `DWORD` |
| Value range | 0 to `EAX_MAX_ENVIRONMENT` |
| Default value | `EAX_ENVIRONMENT_GENERIC` |
| Value units | Integers that each specify a specific environment |

Environment is the fundamental listener property. You typically set it first and then—if you want—modify it using other listener properties described later in this section.

When you set an environment, you choose the acoustic surroundings of the listener—the size of the virtual room around the listener and the reflective qualities of its walls. When you modify the environment using the other listener properties you can, in particular, modify the room size or adjust lower-level parameters such as the overall level and delay of the first reflections, the overall level of the subsequent reverberation, and the duration and tonal quality of the reverberation decay.

To specify an environment, use an integer constant defined in the `eax.h` file. Those environment constants are:

- EAX_ENVIRONMENT_GENERIC
- EAX_ENVIRONMENT_PADDEDCELL
- EAX_ENVIRONMENT_ROOM
- EAX_ENVIRONMENT_BATHROOM
- EAX_ENVIRONMENT_LIVINGROOM
- EAX_ENVIRONMENT_STONEROOM
- EAX_ENVIRONMENT_AUDITORIUM
- EAX_ENVIRONMENT_CONCERTHALL
- EAX_ENVIRONMENT_CAVE
- EAX_ENVIRONMENT_ARENA
- EAX_ENVIRONMENT_HANGAR
- EAX_ENVIRONMENT_CARPETEDHALLWAY
- EAX_ENVIRONMENT_HALLWAY
- EAX_ENVIRONMENT_STONECORRIDOR
- EAX_ENVIRONMENT_ALLEY
- EAX_ENVIRONMENT_FOREST
- EAX_ENVIRONMENT_CITY
- EAX_ENVIRONMENT_MOUNTAINS
- EAX_ENVIRONMENT_QUARRY
- EAX_ENVIRONMENT_PLAIN
- EAX_ENVIRONMENT_PARKINGLOT
- EAX_ENVIRONMENT_SEWERPIPE
- EAX_ENVIRONMENT_UNDERWATER
- EAX_ENVIRONMENT_DRUGGED
- EAX_ENVIRONMENT_DIZZY
- EAX_ENVIRONMENT_PSYCHOTIC

The name of each constant gives you a good idea of the environment's acoustic qualities: small room, large room, live surfaces, dead surfaces, and so forth. EAX_ENVIRONMENT_GENERIC, environment 0, is the default Environment setting. It specifies an average room size with a typical reverberation quality.

Whenever you set the Environment property alone (that is, not as part of a set of all the listener property values at once, defined by DSPROPERTY_EAXLISTENER_ALL), EAX automatically sets the values of all the other listener properties to defaults for the specified environment.

## Environment Size

| | |
|---|---|
| Specify using this ID | DSPROPERTY_EAXLISTENER_ENVIRONMENTSIZE |
| Value type | FLOAT |
| Value range | 1.0 to 100.0 |
| Default value | 7.5 |
| Value units | Linear meters |

The Environment Size property sets the apparent size of the surrounding "room." The value of Environment Size can be considered a characteristic dimension of the room expressed in meters. Scaling Environment Size is equivalent to scaling all dimensions of the room by the same factor.

Because Environment Size is a high-level property, when you change it, it applies a relative adjustment to five lower-level listener properties that determine the shape of the reverberation response: Reflections, Reflections Delay, Reverb, Reverb Delay, and Decay Time. When you change the value of Environment Size, it actually changes the values of the lower-level properties. You can call Get on any of those lower-level properties to see how they have been changed.

Although Environment Size by default affects all five of the lower-level properties mentioned above, you can disable its effect on some or all of these properties by using the Flags listener property described later. It contains one flag for each of the lower-level protocols controlled by Environment Size. All flags are set to TRUE by default to provide a convincing simulation of change in the dimensions of a room (maintaining the reflective properties of its walls). You can override Environment Size's automatic scaling of any of the lower-level protocols by changing its flag to FALSE. You'll find more information about this in the description of the Flags property.

Note that when Environment Size is set to a small value (about two meters or less), it adds a coloration effect characteristic of small rooms to the reverberation. This effect becomes stronger as Environment Size is reduced.

## Environment Diffusion

| | |
|---|---|
| Specify using this ID | DSPROPERTY_EAXLISTENER_ ENVIRONMENTDIFFUSION |
| Value type | FLOAT |
| Value range | 0.0 to 1.0 |
| Default value | Varies depending on the environment |
| Value units | A linear multiplier value |

The Environment Diffusion property controls the echo density in the reverberation decay. It's set by default to 1.0, which provides the highest density. Reducing diffusion gives the reverberation a more "grainy" character that is especially noticeable with percussive sound sources. If you set a diffusion value of 0.0, the later reverberation sounds like a succession of distinct echoes.

## Room and Room HF

| | |
|---|---|
| Specify using this ID | DSPROPERTY_EAXLISTENER_ROOM |
| Value type | LONG |
| Value range | -10000 to 0 |
| Default value | Varies depending on the environment |
| Value units | Hundredths of a dB |

The Room property is the master volume control for the reflected sound (both early reflections and reverberation) that EAX adds to all sound sources. It sets the maximum amount of reflections and reverberation added to the sound mix in the primary buffer (the listener). The value of the Room property ranges from 0 dB (the maximum amount) to -100 dB (no reflected sound at all).

| | |
|---|---|
| Specify using this ID | DSPROPERTY_EAXLISTENER_ROOMHF |
| Value type | LONG |
| Value range | -10000 to 0 |
| Default value | Varies depending on the environment |
| Value units | Hundredths of a dB |

The Room HF property further tweaks reflected sound by attenuating it at high frequencies. It controls a low-pass filter that applies globally to the reflected sound of all sound sources. The value of the Room HF property ranges from 0 dB (no filter) to -100 dB (virtually no reflected sound).

Although the amount and quality of reflected sound controlled by Room and Room HF is global for all sound sources, you can vary the reflected sound for individual sources by setting each source's corresponding sound-source properties—also called Room and Room HF. These source properties treat the listener Room and Room HF properties as a baseline, and are added to the baseline value to determine the final amount of reflected sound for each sound source. (For more information see the description of Room and Room HF in the section on sound-source properties.)

## Decay Time and Decay HF Ratio

| | |
|---|---|
| Specify using this ID | DSPROPERTY_EAXLISTENER_DECAYTIME |
| Value type | FLOAT |
| Value range | 0.1 to 20.0 |
| Default value | Varies depending on the environment |
| Value units | Seconds |

The Decay Time property sets the reverberation decay time. It ranges from 0.1 (typically a small room with very dead surfaces) to 20.0 (typically a large room with very live surfaces). This low-level property may be controlled by the high-level listener property Environment Size, in which case its value is scaled according to the value set there. You can disable that automatic scaling by setting the appropriate flag in the listener property Flags. (See its description for more information.)

| | |
|---|---|
| Specify using this ID | DSPROPERTY_EAXLISTENER_DECAYTIMEHFRATIO |
| Value type | FLOAT |
| Value range | 0.1 to 20.0 |
| Default value | Varies depending on the environment |
| Value units | A linear multiplier value |

The Decay HF Ratio property sets the spectral quality of the Decay Time parameter. It is the ratio of high-frequency decay time relative to the time set by Decay Time. The Decay HF Ratio value 1.0 is neutral: the decay time is equal for all frequencies. As Decay HF Ratio increases above 1.0, the high-frequency decay time increases so it's longer than the decay time at low frequencies. You hear a more brilliant reverberation with a longer decay at high frequencies. As the Decay HF Ratio value decreases below 1.0, the high-frequency decay time decreases so it's shorter than the decay time of the low frequencies. You hear a more natural reverberation.

## Reflections and Reflections Delay

| | |
|---|---|
| Specify using this ID | DSPROPERTY_EAXLISTENER_REFLECTIONS |
| Value type | LONG |
| Value range | -10000 to 1000 |
| Default value | Varies depending on the environment |
| Value units | Hundredths of a dB |

The Reflections property controls the overall amount of initial reflections relative to the Room property. (The Room property sets the overall amount of reflected sound: both initial reflections and later reverberation.) The value of Reflections ranges from a maximum of 10 dB to a minimum of -100 dB (no initial reflections at all), and is corrected by the value of the Room property. The Reflections property does not affect the subsequent reverberation decay.

You can increase the amount of initial reflections to simulate a more narrow space or closer walls, especially effective if you associate the initial reflections increase with a reduction in reflections delays by lowering the value of the Reflection Delay property. To simulate open or semi-open environments, you can maintain the amount of early reflections while reducing the value of the Reverb property, which controls later reflections.

| | |
|---|---|
| Specify using this ID | DSPROPERTY_EAXLISTENER_REFLECTIONSDELAY |
| Value type | FLOAT |
| Value range | 0.0 to 0.3 |
| Default value | Varies depending on the environment |
| Value units | Seconds |

The Reflections Delay property is the amount of delay between the arrival time of the direct path from the source to the first reflection from the source. It ranges from 0 to 300 milliseconds. You can reduce or increase Reflections Delay to simulate closer or more distant reflective surfaces—and therefore control the perceived size of the room.

Both Reflections and Reflections Delay are low-level properties that may be controlled by the high-level listener property Environment Size. If so, their values are scaled according to the value set there. You can disable Environment Size's control over these properties by setting the appropriate flags in the listener property Flags. (See its description for more information.)

## Reverb and Reverb Delay

| | |
|---|---|
| Specify using this ID | DSPROPERTY_EAXLISTENER_REVERB |
| Value type | LONG |
| Value range | -10000 to 2000 |
| Default value | Varies depending on the environment |
| Value units | Hundredths of a dB |

The Reverb property controls the overall amount of later reverberation relative to the Room property. (The Room property sets the overall amount of both initial reflections and later reverberation.) The value of Reverb ranges from a maximum of 20 dB to a minimum of -100 dB (no late reverberation at all).

Note that Reverb and Decay Time are independent properties: If you adjust Decay Time without changing Reverb, the total intensity (the averaged square of the amplitude) of the late reverberation remains constant.

| | |
|---|---|
| Specify using this ID | DSPROPERTY_EAXLISTENER_REVERBDELAY |
| Value type | FLOAT |
| Value range | 0.0 to 0.1 |
| Default value | Varies depending on the environment |
| Value units | Seconds |

The Reverb Delay property defines the begin time of the late reverberation relative to the time of the initial reflection (the first of the early reflections). It ranges from 0 to 100 milliseconds. Reducing or increasing Reverb Delay is useful for simulating a smaller or larger room.

Both Reverb and Reverb Delay are low-level properties that may be controlled by the high-level listener property Environment Size. If so, their values are scaled according to the value set there. For example, if you increase Environment Size, the intensity of later reverberation reduces. You can disable Environment Size's automatic scaling of these properties by setting the appropriate flags in the listener property Flags. (See its description for more information.)

## Room Rolloff Factor

| | |
|---|---|
| Specify using this ID | DSPROPERTY_EAXLISTENER_ ROOMROLLOFFFACTOR |
| Value type | FLOAT |
| Value range | 0.0 to 10.0 |
| Default value | 0.0 |
| Value units | A linear multiplier value |

The Room Rolloff property is one of two methods available in EAX to attenuate the reflected sound (containing both reflections and reverberation) according to source-listener distance. It's defined the same way as DirectSound's Rolloff Factor, but operates on reflected sound instead of direct-path sound. Setting the Room Rolloff Factor value to 1.0 specifies that the reflected sound will decay by 6 dB every time the distance doubles. Any value other than 1.0 is equivalent to a scaling factor applied to the quantity specified by ((Source listener distance) - (Minimum Distance)). Minimum distance is a DirectSound sound-source parameter that specifies the inner border for distance rolloff effects: if the source comes closer to the listener than the minimum distance, the direct-path sound isn't increased as the source comes closer to the listener, and neither is the reflected sound.

The default value of Room Rolloff Factor is 0.0 because, by default, EAX naturally manages the reflected sound level automatically for each sound source to simulate the natural rolloff of reflected sound vs. distance in typical rooms. (Note that this isn't the case if the source property flag Reverb Auto is set to false.) You can use Room Rolloff Factor as an option to automatic control so you can exaggerate or replace the default automatically-controlled rolloff.

## Air Absorption HF

| | |
|---|---|
| Specify using this ID | DSPROPERTY_EAXLISTENER_AIRABSORPTIONHF |
| Value type | FLOAT |
| Value range | -100.0 to 0.0 |
| Default value | -5.0 |
| Value units | Hundredths of a dB per meter |

The Air Absorption HF property controls the distance-dependent attenuation at high frequencies caused by the propagation medium. It applies to both the direct path and reflected sound. You can use Air Absorption HF to simulate sound transmission through foggy air, dry air, smoky atmosphere, and so on. The default value is -0.05 dB per meter, which roughly corresponds to typical condition of atmospheric humidity, temperature, and so on. Lowering the value simulates a more absorbent medium (more humidity in the air, for example); raising the value simulates a less absorbent medium (dry desert air, for example).

## Flags

| | |
|---|---|
| Specify using this ID | DSPROPERTY_EAXLISTENER_FLAGS |
| Value type | DWORD |
| Value range | 0x00000000 to 0x0000002F |
| Default value | Varies depending on the environment |
| Value units | Each bit of a value is a Boolean flag |

The Flags property uses its six low-order bits to set six listener-property flags. Five of these flags, the "scale flags," set the control that listener property Environment Size has over five lower-level listener properties. The sixth flag prevents excessive decay times at high frequencies.

The scale flags are all set to TRUE by default so Environment Size has control over the lower-level properties to simulate the typical behavior expected for a change of room dimensions. The sixth flag, Decay HF Limit, is set by the property Environment, and may be true or false depending on the specified environment.

Each flag is represented by a constant in the eax.h file. To set the flag true, call Get on the listener interface to get the current value of the Flags property. Bitwise OR the flag constant with the value of the Flags property, then call Set on the listener interface using the revised Flags value. To set the flag false, Get and Set the value as before, but instead of using bitwise OR, first NOT the flag constant then AND it with the retrieved value.

The flags and their constants are these:

### *Decay Time Scale*

Constant: EAXLISTENERFLAGS_DECAYTIMESCALE

If this flag is TRUE, a change in Environment Size value causes a proportional change of the property Decay Time. If it's FALSE, a change in Environment Size has no effect on Decay Time.

### *Reflections Delay Scale*

Constant: EAXLISTENERFLAGS_REFLECTIONSSCALE

If this flag is TRUE, a change in Environment Size value causes a proportional change of the property Reflections Delay. (In effect, as the room gets larger the nearest walls get more distant.) If it's FALSE, a change in Environment Size has no effect on Reflections Delay. (In effect, as the room gets larger or smaller the nearest walls stay the same distance.)

### *Reverb Delay Scale*

Constant: EAXLISTENERFLAGS_REFLECTIONSDELAYSCALE

If this flag is TRUE, a change in Environment Size value causes a proportional change of the property Reverb Delay. If it's FALSE, a change in Environment Size has no effect on Reverb Delay.

### *Reflections Scale*

Constant: EAXLISTENERFLAGS_REVERBSCALE

If both this flag and the Reflections Delay Scale flag are TRUE, an increase in Environment Size value causes an attenuation of the property Reflections. If it's FALSE, a change in Environment Size has no effect on Reflections.

### *Reverb Scale*

Constant: EAXLISTENERFLAGS_REVERBDELAYSCALE

If this flag is TRUE, an increase in Environment Size value causes an attenuation of the property Reverb. If it's FALSE, a change in Environment Size has no effect on Reverb.

### *Decay HF Limit*

Constant: EAXLISTENERFLAGS_DECAYHFLIMIT

If this flag is TRUE, high-frequency decay time automatically stays below a limit value that's derived from the setting of the property Air Absorption HF. This limit applies regardless of the setting of the property Decay HF Ratio, and the limit doesn't affect the value of Decay HF Ratio. This limit, when on, maintains a natural sounding reverberation decay by allowing you to increase the value of Decay Time without the risk of getting an unnaturally long decay time at high frequencies.

If this flag is FALSE, high-frequency decay time isn't automatically limited.

## THE SOUND-SOURCE PROPERTY SET

The sound-source property set contains thirteen properties that apply through a property-set interface to a particular secondary buffer (which constitutes a sound source). To use these properties, you must specify the property-set GUID DSPROPSETID_EA2_BufferProperties.

### Direct and Direct HF

| | |
|---|---|
| Specify using this ID | DSPROPERTY_EAXBUFFER_DIRECT |
| Value type | LONG |
| Value range | -10000 to 1000 |
| Default value | 0 |
| Value units | Hundredths of a dB |

The Direct property is a low-level property that applies a relative correction to this sound source's direct-path intensity. (Direct-path intensity is the level of the sound source after attenuation for distance, orientation, and so on.) The Direct property allows you to apply a manual correction in addition to the effect of the DirectSound parameters distance, rolloff factor, orientation, and cone angles and to the effect of other EAX sound-source properties described in this section. The default value of 0 adds no correction to direct-path sound.

If you specify an increase in direct-path sound that, when combined with the direct-path sound set by both DirectSound and EAX properties results in a total gain larger than the intensity of the unattenuated sound source (referred to as 0 dB), the EAX driver clips the resulting send level to 0 dB. This means that the Direct property can only increase direct-path sound that has been attenuated below the original source sound by these other factors. And that it can never increase the direct-path sound to more than the original sound source.

| | |
|---|---|
| Specify using this ID | DSPROPERTY_EAXBUFFER_DIRECTHF |
| Value type | LONG |
| Value range | -10000 to 0 |
| Default value | 0 |
| Value units | Hundredths of a dB |

The Direct HF property is a related low-level property that applies a relative correction to the high-frequency component of the sound source's direct-path intensity. It has the same relationship to the high-frequency direct-path components of other DirectSound and EAX properties that the Direct property has to the full-frequency direct-path intensity.

## Room and Room HF

| | |
|---|---|
| Specify using this ID | DSPROPERTY_EAXBUFFER_ROOM |
| Value type | LONG |
| Value range | -10000 to 1000 |
| Default value | 0 |
| Value units | Hundredths of a dB |

The Room property is a low-level sound-source property that's defined the same way as the listener (global) Room property: it is the volume control for reflected sound (early reflections and reverberation). In this case, however, Room applies only to this sound source, and therefore affects only the reflected sound added to this source. It is an additive property; its setting is added to the total reflected sound value for this source that is specified by all other EAX listener and source properties.

You can use the Room sound-source property to correct the intensity of reflected sound at minimum distance as defined by the Room listener property. This is useful especially if different sound sources have different minimum distances.

Note that although you can specify a positive Room value, the combined reflected-sound volume cannot go above 0 dB—that is, it can never be louder than the volume of the sound source without attenuation. You can use positive values to restore volume that has been diminished by other properties, but you can never go beyond that.

| | |
|---|---|
| Specify using this ID | DSPROPERTY_EAXBUFFER_ROOMHF |
| Value type | LONG |
| Value range | -10000 to 0 |
| Default value | 0 |
| Value units | Hundredths of a dB |

The Room HF property is a low-level sound-source property that is defined the same way as the listener Room HF property, but in this case is applied only to this sound source. It controls a low-pass filter that applies to the reflected sound added to this source. The value of Room HF ranges from 0 dB (no filter) to -100 dB (virtually no reflected sound). This is an additive property; its setting is added to the listener Room HF property and the other sound-source properties to determine how filtered the reflected sound for this source.

## Obstruction and Obstruction LF Ratio

| | |
|---|---|
| Specify using this ID | DSPROPERTY_EAXBUFFER_OBSTRUCTION |
| Value type | LONG |
| Value range | -10000 to 0 |
| Default value | 0 |
| Value units | Hundredths of a dB |

The Obstruction property specifies the amount of obstruction muffling to apply to a sound source's direct-path sound. Obstruction occurs when an object lies between a sound source and a listener who occupy the same room. The direct path from source to listener is muffled by the obstruction, but the reflected sound from the source remains unchanged.

You can set the value of the Obstruction property for each sound source. The value controls the degree to which the direct path from the source is muffled. If the Obstruction LF Ratio property (described below) is set to 0.0, Obstruction controls only attenuation at high frequencies. If Obstruction LF Ratio is set above 0.0, Obstruction also muffles low frequencies to the extent specified by Obstruction LF Ratio.

Obstruction's maximum value, 0, specifies no attenuation and hence no obstruction effect. The minimum value, –10000 (which is -100 dB), indicates that the sound source is so obstructed that the direct path from source to listener is negligible—so only the source's reflected sound is audible. Any value between minimum and maximum indicates partial obstruction.

Note that you can use Obstruction and Occlusion simultaneously. If, for example, the source is in another room from the listener and there's also a large obstacle between the listener and the wall. In this case the dry path is filtered twice: once by Occlusion and once by Obstruction.

| | |
|---|---|
| Specify using this ID | DSPROPERTY_EAXBUFFER_OBSTRUCTIONLFRATIO |
| Value type | FLOAT |
| Value range | 0.0 to 1.0 |
| Default value | 0.0 |
| Value units | A linear multiplier value |

The Obstruction LF Ratio property affects the spectral quality of obstruction set by the Obstruction property: it specifies the obstruction attenuation at low frequencies relative to the attenuation at high frequencies. The minimum value of 0.0 (the default value) specifies no attenuation at low frequencies; the maximum value of 1.0 specifies the same low-frequency attenuation as high-frequency attenuation. Note that adjusting Obstruction LF Ratio alone has no effect if Obstruction is set to 0.

### *Material Presets*

There are times when an obstructing object can transmit direct sound as well as diffract it. If so, you can use the obstruction properties to specify that transmission quality of the object. The most effective way to do this is to use a material preset. You can read about material presets in Appendix C.

## Occlusion, Occlusion LF Ratio, and Occlusion Room Ratio

| | |
|---|---|
| Specify using this ID | DSPROPERTY_EAXBUFFER_OCCLUSION |
| Value type | LONG |
| Value range | -10000 to 0 |
| Default value | 0 |
| Value units | Hundredths of a dB |

The Occlusion property specifies the amount of occlusion muffling to apply to a sound source's direct sound and to its reflected sound. Occlusion occurs when the listener is in one room or environment, the sound source is in another room or environment, and the listener hears the sound through a separating wall or through an opened or closed door or window. Both the direct sound and the reflected sound from the sound source are muffled by the occlusion.

The effect of occlusion depends a great deal on the sound transmission qualities of the material separating the two rooms. Some materials are thick and absorbent and transmit very little sound; others are stiff and thin and transmit clearly; others have transmission qualities in between. Frequency response varies too. Some materials attenuate high frequencies more than others do.

You can set the value of the Occlusion property for each sound source. The value controls the degree to which both the direct-path and reflected sound from the source is muffled. If the Occlusion LF Ratio property (described below) is set to 0.0, Occlusion controls only attenuation at high frequencies. If Occlusion LF Ratio is set above 0.0, Occlusion also muffles low frequencies to the extent specified by Occlusion LF Ratio.

Occlusion's maximum value, 0, specifies no attenuation and hence no occlusion effect. The minimum value, –10000 (which is -100 dB), indicates that the sound source is so occluded that it is completely inaudible—at least if Occlusion LF Ratio specifies that low frequencies are equally occluded with high frequencies. If it doesn't, low frequencies will still be audible. Any value between minimum and maximum indicates partial occlusion.

| | |
|---|---|
| Specify using this ID | DSPROPERTY_EAXBUFFER_OCCLUSIONLFRATIO |
| Value type | FLOAT |
| Value range | 0.0 to 1.0 |
| Default value | 0.25 |
| Value units | A linear multiplier value |

The Occlusion LF Ratio property affects the spectral quality of occlusion set by the Occlusion property: it specifies the occlusion attenuation at low frequencies relative to the attenuation at high frequencies. The minimum value of 0.0 specifies no attenuation at low frequencies; the maximum value of 1.0 specifies the same low-frequency attenuation as high-frequency attenuation. The default setting of 0.25 specifies that low frequencies are attenuated much less than high frequencies. Note that adjusting Occlusion LF Ratio alone has no effect if Occlusion is set to 0.

| | |
|---|---|
| Specify using this ID | DSPROPERTY_EAXBUFFER_OCCLUSIONROOMRATIO |
| Value type | FLOAT |
| Value range | 0.0 to 10.0 |
| Default value | 0.5 |
| Value units | A linear multiplier value |

The Occlusion Room Ratio property specifies the additional amount of occlusion attenuation to be applied to reflected sound (early reflections and reverberation). The minimum value of 0.0 specifies no additional reflected sound occlusion attenuation—that is, that direct-path and reflected sound attenuation occurs equally in the amount set by the Occlusion property. The maximum value of 10.0 specifies that the reflected sound is equal to 10 times the setting of Occlusion.

The default value of 0.5 specifies that reflected sound undergoes an additional attenuation that is half the setting of Occlusion. This creates a natural sensation of occlusion because in the physical world, the occluding wall acts as a secondary sound source in the listener's room. Because the wall radiates sound in only half the directions that a sound source in the middle of the room can, it generates significantly less reflected sound than the original source would be if it were located in the room.

### *Material Presets*

An effective way to set all three occlusion properties at once is to use a material preset. A material preset is an array of three occlusion settings chosen to represent the sound transmission properties of a specific occluding material. Appendix C tells you how to use material presets.

## Room Rolloff Factor

| | |
|---|---|
| Specify using this ID | DSPROPERTY_EAXBUFFER_ROOMROLLOFFFACTOR |
| Value type | FLOAT |
| Value range | 0.0 to 10.0 |
| Default value | 0.0 |
| Value units | A linear multiplier value |

The Room Rolloff Factor property is a low-level sound-source property that is defined the same way as the listener (global) Room Rolloff property: it is one of two methods available in EAX to attenuate the reflected sound (early reflections and reverberation) according to source-listener distance. In this case, however, Room Rolloff applies only to this sound source, and therefore affects only the reflected sound generated by this source. Room Rolloff is an additive property; its setting is added to the listener Room Rolloff setting to get the final room rolloff multiplier value for that source.

The resulting combined multiplier value affects the amount of room rolloff. A value of 1.0 specifies that the reflected sound will decay by 6 dB every time the distance doubles. Any value other than 1.0 is equivalent to a scaling factor that's applied to the quantity ((source listener distance) - (Minimum Distance)).

## Air Absorption Factor

| | |
|---|---|
| Specify using this ID | DSPROPERTY_EAXBUFFER_<br>AIRABSORPTIONFACTOR |
| Value type | FLOAT |
| Value range | 0.0 to 10.0 |
| Default value | 1.0 |
| Value units | A linear multiplier value |

The Air Absorption Factor property is a multiplier value for the air absorption value set by the listener property Air Absorption HF. The resultant air absorption value applies only to this sound source.

The air absorption value controls the distance-dependent attenuation at high frequencies caused by the propagation medium. It applies to both the direct-path and reflected sound, and can simulate sound transmission through foggy air, dry air, smoky atmosphere, and so on. The Air Absorption Factor default value of 1.0 specifies no change to the air absorption factor set by the listener Air Absorption HF property. The minimum value of 0.0 turns off air absorption for this source and a maximum value of 10.0 multiplies absorption by 10 for this source.

You can use the Air Absorption Factor to simulate a source located in different atmospheric conditions than the rest of the room. You can increase air absorption, for example, for a sound source that comes from the middle of a cloud of smoke. Or you can decrease air absorption for a sound source coming from a suddenly visible object in moving clouds.

## Outside Volume HF

| | |
|---|---|
| Specify using this ID | DSPROPERTY_EAXBUFFER_OUTSIDEVOLUMEHF |
| Value type | LONG |
| Value range | -10000 to 0 |
| Default value | 0 |
| Value units | Hundredths of a dB |

The Outside Volume HF property enhances the directivity effect provided by DirectSound 3D for individual sound sources. A directed sound source points in a specified direction. The source sounds at full volume when the listener is directly in front of the source; it's attenuated as the listener circles the source away from the front.

When DirectSound attenuates a source's direct-path sound to simulate directivity, it attenuates high- and low-frequency sounds equally. Real world sources tend to be more directive at high frequencies than at low frequencies.

The Outside Volume HF property enhances DirectSound's directivity effect at your option by attenuating the high frequencies more than its low frequencies in the rear of the source. At the minimum (and default) setting of 0, there is no additional high-frequency attenuation, so DirectSound's directivity effect is unaltered. At the maximum setting of –10000 (-100 dB), directivity attenuation for high frequencies is 100 dB more than it is for low frequencies.

This property sets directivity high-frequency attenuation for both the direct-path and the reflected sounds of the sound source. You can turn off its effect on direct-path sound using the Direct HF Auto flag, or you can turn off its effect on reflected sound using the Room HF Auto flag. Both flags are described later under the sound-source property Flags.

Note that if you use Outside Volume HF systematically on all sources and have Room HF Auto turned on, it may sound more natural to set the listener property Room HF to 0 (or raise it closer to 0) on all environment presets. If you have listener Room HF set far below zero, then you apply a low-pass filter to sound source's already affected by Outside Volume HF's low pass filtering

## Flags

| | |
|---|---|
| Specify using this ID | DSPROPERTY_EAXBUFFER_FLAGS |
| Value type | DWORD |
| Value range | 0x00000000 to 0x00000007 |
| Default value | 0x00000007 (all flags are set true) |
| Value units | Each bit of a value is a Boolean flag |

The Flags property uses its three low-order bits to set three sound-source-property flags. These flags determine whether or not you want the EAX engine to automatically adjust the intensity and tonal color of the reverberation for a source according to the setting of DirectSound's position and directivity parameters. These flags are all set to TRUE by default to provide a more realistic experience without any programming work.

Each flag is represented by a constant in the eax.h file. To set the flag true, call Get on the listener interface to get the current value of the Flags property. Bitwise OR the flag constant with the value of the Flags property, then call Set on the listener interface using the revised Flags value. To set the flag false, Get and Set the value as before, but instead of using bitwise OR, first NOT the flag constant then AND it with the retrieved value.

The flags and their constants are these:

### *Direct HF Auto*

Constant: EAXBUFFERFLAGS_DIRECTHFAUTO

If this flag is TRUE (its default value), this sound source's direct-path sound is automatically filtered according to the orientation of the source relative to the listener and the setting of the sound-source property Outside Volume HF. If Outside Volume HF is set to 0, the source is not more directive at high frequencies and this flag has no effect. Otherwise, the direct path will be brighter in front of the source than on the side on in the rear.

If this flag is FALSE, this sound source's direct-path sound isn't filtered at all according to orientation. Note that this isn't the same as setting Outside Volume to 0, because this flag doesn't affect high-frequency attenuation of each source's *reflected sound* in response to DirectSound's directivity attenuation. That's controlled by the Room HF Auto flag, described later.

### *Room HF Auto*

Constant: EAXBUFFERFLAGS_ROOMHFAUTO

If this flag is TRUE (its default value), the intensity of this sound source's reflected sound at high frequencies will be automatically attenuated according to the high-frequency source directivity as set by the EAX Outside Volume HF property. If Outside Volume HF is set to 0, the source is not more directive at high frequencies and this flag has no effect. Otherwise, making the source more directive at high frequencies will have the natural effect of reducing the amount of high frequencies in the reflected sound.

If this flag is FALSE, the sound source's reflected sound isn't filtered at all according to the source's directivity. Note that this isn't the same as setting Outside Volume to 0, because this flag doesn't affect high-frequency attenuation of each source's *direct-path sound* in response to DirectSound's directivity attenuation. That's controlled by the Direct HF Auto flag, described earlier.

### *Room Auto*

Constant: EAXBUFFERFLAGS_ROOMAUTO

If this flag is TRUE (its default value), the intensity of this sound source's reflected sound is automatically attenuated according to source-listener distance and source directivity (as determined by DirectSound's Inside Cone Angle, Outside Cone Angle, and Outside Cone Volume parameters). If it's FALSE, the reflected sound isn't attenuated according to distance and directivity.

# APPENDIX A: CREATING AN EAX OBJECT IN DIRECTSOUND

The global reverb object (the EAX listener interface object) is created by calling the QueryInterface function on a secondary 3D buffer (a sound-source buffer) even though using that object will affect all sound sources. That's because DirectSound doesn't allow a property set interface for the listener buffer. This may cause a potential problem if EAX settings are asserted after the secondary buffer has been released. To prevent this, re-establish the EAX listener object through another secondary buffer. You may be wise to create a small dummy secondary buffer and keep that buffer open for the life of the game. The following sample code shows you how to create the listener object with this dummy buffer in mind:

```
#define PSET_SETGET (KSPORPERTY_SUPPORT_GET|KSPROPERTY_SUPPORT_SET)

//  local small dummy buffer
    static LPDIRECTSOUNDBUFFER pBuffer=NULL;     // dummy DS buffer

//--------------------------------------------------------------------
//
    //
    //  EAXCreate
    //
    //  DESCRIPTION:
    //    Opens an EAX environment if one is available
    //
    //  PARAMETERS:
    //    pDS: direct sound object
    //
    //  RETURNS:
    //    EAX object pointer, NULL if none can be found
    //--------------------------------------------------------------------
----//

LPKSPROPERTYSET EAXCreate(LPDIRECTSOUND pDS)
      {
      WAVEFORMATEX fmt={WAVE_FORMAT_PCM,2,44100,176400,4,16,0};

    //  for dummy 3D buffer
      DSBUFFERDESC desc;
      LPDIRECTSOUND3DBUFFER p3DBuf;
                                  //  3D buffer interface
      LPKSPROPERTYSET pEAX;
                          //  property set interface
      unsigned long support=0;
                          //  variable to hold support status

  //  Make sure previous dummy buffer has been released
```

```
      if (pBuffer)
        {
        IDirectSoundBuffer_Release(pBuffer);
        pBuffer=NULL;
        }

  //  Create the dummy buffer
      memset(&desc, 0, sizeof(DSBUFFERDESC));
      desc.dwSize = sizeof(DSBUFFERDESC);
      desc.dwFlags = DSBCAPS-STATIC | DSBCAPS_CTRL3D;
      desc.dwBufferBytes = 128;
      desc.lpwfxFormat = &fmt;
      if (IDirectSound_CreateSoundBuffer(pDS,&desc,&pBuffer,NULL) !=

           DS_OK)
         return NULL;

  //  Create a 3D buffer interface
      if (IDirectSoundBuffer_QueryInterface(pBuffer,
         &IID_IDirectSound3DBuffer,&p3DBuf) != DS_OK
        ||
IDirectSound3DBuffer_QueryInterface(p3DBuf,&IID_IKsPropertySet,
          &pEAX) != DS_OK)
    {
        IDirectSoundBuffer_Release(pBuffer);
        pBuffer=NULL;
        return NULL;
        }

  //  Create the EAX reverb interface
      if
(IKsPropertySet_QuerySupport(pEAX,&DSPROPSETID_EAX_ListenerProperties,
     DSPROPERTY_EAXLISTENER_ALL,&support) != DS_OK
        || (support & PSET_SETGET) != PSET_SETGET)
        {
        IDirectSoundBuffer_Release(pBuffer);
        pBuffer=NULL;
        pEAX=NULL;
        }

  return pEAX;
      }

//------------------------------------------------------------------------
//
    //
    //  EAXrelease
    //
    //  DESCRIPTION:
    //    Closes an EAX environment
    //
    //  PARAMETERS:
    //    pEAX: EAX object
```

```
    //
    //  RETURNS:
    //    EAX object pointer, NULL if none can be found
    //------------------------------------------------------------//

void EAXRelease(LPKSPROPERTYSET pEAX)
      {
      if (pEAX)
        IKsPropertySet_Release(pEAX);
      if (pBuffer)
        {
        IDirectSoundBuffer_Release(pBuffer);
        pBuffer=NULL;
        }

      }
```

# APPENDIX B: DEFERRED SETTINGS

EAX allows you to postpone when property settings are executed. The main reason to do this is to save CPU resources. In certain cases it can also avoid audio artifacts, such as popping sounds, that can result from temporary combinations of settings during the execution of your program.

As you set properties in the EAX interface of the listener or of any sound source, you can specify that the settings be deferred. They won't be executed until you later set a property there using immediate execution, or you directly specify that all deferred settings be executed. At that point they're executed simultaneously.

## DEFERRING SETTINGS

To defer a listener property setting, call `set` on the property value and bitwise `OR` the property identifier with `DSPROPERTY_EAXLISTENER_DEFERRED` before (or as) you pass the identifier as an argument. The example below shows how:

```
In DirectSound:

LONG Reverb=-600;

pReverb->Set(DSPROPSETID_EAX_ListenerProperties,
    DSPROPERTY_EAXLISTENER_REVERB |
    DSPROPERTY_EAXLISTENER_DEFERRED,NULL,0,&Reverb,sizeof(LONG));


In OpenAL:

LONG Reverb=-600;

pfPropSet(PROPSETID_EAX_ListenerProperties,
    DSPROPERTY_EAXLISTENER_REVERB | DSPROPERTY_EAXLISTENER_DEFERRED, 0,
    &Reverb, sizeof(LONG));
```

The code below shows a similar example by making a deferred setting to the Room property on a particular sound source:

```
In DirectSound:

LONG Room=0;
pReverbBuffer[i]->Set(DSPROPSETID_EAX_BufferProperties,
    DSPROPERTY_EAXBUFFER_ROOM|DSPROPERTY_EAXLISTENER_DEFERRED,
    NULL,0,&Room,sizeof(LONG));


In OpenAL:

LONG Room=0;
pfPropSet(PROPSETID_EAX_SourceProperties, DS_PROPERTY_EAXBUFFER_ROOM |
    DSPROPERTY_EAXLISTENER_DEFERRED, NULL, 0, &Room, sizeof(LONG);
```

## EXECUTING DEFERRED SETTINGS

Any time you set a property value and don't specify that it be deferred, which is the "immediate" (and default) mode of property setting, you ask the interface to execute the setting immediately along with any other deferred settings it's holding. If you want to make your code explicit, you can directly specify immediate mode by bitwise ORing the property identifier with DSPROPERTY_EAXLISTENER_IMMEDIATE as shown in this code example:

```
In DirectSound:

LONG Reverb=-600;
pReverb->Set(DSPROPSETID_EAX_ListenerProperties,
    DSPROPERTY_EAXLISTENER_REVERB|DSPROPERTY_EAXLISTENER_IMMEDIATE,
    NULL,0,&Reverb,sizeof(LONG));

In OpenAL:

LONG Reverb=-600;
pfPropSet((PROPSETID_EAX_ListenerProperties,
    DSPROPERTY_EAXLISTENER_REVERB|DSPROPERTY_EAXLISTENER_IMMEDIATE,
    NULL,0,&Reverb,sizeof(LONG));
```

If you want to execute deferred settings without setting a value, you can use a special property identifier: DSPROPERTY_EAXLISTENER_COMMITDEFERREDSETTINGS. The example below shows how:

```
In DirectSound:

LONG Reverb=-600;
pReverb->Set(DSPROPSETID_EAX_ListenerProperties,
    DSPROPERTY_EAXLISTENER_COMMITDEFERREDSETTINGS,
    NULL,0,NULL,0);

In OpenAL:

LONG Reverb=-600;
pfPropSet(PROPSETID_EAX_ListenerProperties,
    DSPROPERTY_EAXLISTENER_COMMITDEFERREDSETTINGS,
    NULL,0,NULL,0);
```

Note that committing deferred settings using this property identifier will execute any deferred settings for the listener or sound source as well. In DirectSound, if you call DirectSound 3D's `CommitDeferredSettings()`, it executes all EAX deferred settings for the listener or sound source as well as the DirectSound deferred settings.

# APPENDIX C: MATERIAL PRESETS

The EAX SDK includes a set of predefined material presets to help you set occlusion properties or—when an acoustically transmissive object stands between a sound source and the listener—obstruction properties. This appendix describes how to use material presets and how to define your own material presets.

## MATERIAL PRESET COMPONENTS

A material preset is an array of either occlusion values or obstruction values. The array defines the acoustic transmission qualities of the wall or object between the sound source and the listener.

A material preset for occlusion is an array of three values:

- An Occlusion setting
- An Occlusion LF Ratio setting
- An Occlusion Room Ratio setting

A material preset for obstruction is an array of two values:

- An Obstruction setting
- An Obstruction LF Ratio setting

Occlusion and Occlusion LF Ratio take the same range of values as Obstruction and Obstruction LF Ratio. And both pairs of properties define attenuation in the same way—the only difference is that occlusion applies to direct-path and reflected sound while obstruction applies only to direct-path sound. Because of these similarities, a material preset whose values are defined for occlusion will apply just as well for obstruction. The third value in the array (Occlusion Room Ratio) is simply ignored.

## APPLYING A MATERIAL PRESET

To apply a material preset for occlusion, you set the occluded sound source's three occlusion properties to the values in the material preset. If the sound wasn't occluded previously, set Occlusion LF Ratio and Occlusion Room Ratio first, which have no effect as long as Occlusion is set to 0. Set Occlusion last.

If the sound was already occluded using different occlusion settings, the same strategy should work as long as Occlusion is set immediately after the other two properties. (You can also used deferred settings, described in Appendix B, to avoid any sound artifacts when you apply a material preset.)

Material presets for obstruction are usually irrelevant because the source's direct-path sound is transmitted around the obstacle through diffraction. It's only the relative positions of the source, the listener, and the obstacle that should determine the settings of the two obstruction properties. If, however, the sound can be transmitted *through* the obstacle, you can assign a material preset to that obstacle just as you would for an occluding wall.

To apply a material preset for obstruction, you set the obstructed sound source's two obstruction properties to the first two values in the material preset. Ignore the third value, Room Ratio, which doesn't exist for obstruction. As you would for occlusion, you should set the Obstruction value after the Obstruction LF Ratio value.

## PREDEFINED MATERIAL PRESETS

EAX 2.0 comes with eight predefined material presets. You can find them in one of the EAX SDK header files.

## DESIGNING YOUR OWN MATERIAL PRESETS

The following rules of thumb may help you listen to and design your own material presets:

1. Turn the reverberation off in your application by setting the Room property to -100.

5. Adjust the Occlusion and Occlusion LF Ratio properties by ear.

6. Set Occlusion Room Ratio so that Occlusion * ((Occlusion LF Ratio) + (Occlusion Room Ratio)) = -1200.

7. Turn the reverberation back on and adjust Occlusion Room Ratio to get the desired amount of reverberation. (In order to convincingly simulate a source located in the next room, the amount of reverberation that this source generates in the listener's room shouldn't be too high.)