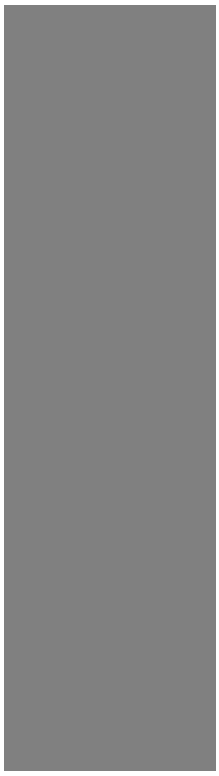
A3D 3.0 API Reference Guide



AUREAL

Disclaimer

This document may not, in whole or part, be copied, reproduced, reduced, or translated by any means, either mechanical or electronic, without prior consent in writing from Aureal Inc. The information in this document has been carefully checked and is believed to be accurate. However, Aureal Inc. assumes no responsibility for any inaccuracies that may be contained in this manual. In no event will Aureal Inc. be liable for direct, indirect, special, incidental, or consequential damages resulting from any defect or omission in this manual, even if advised of the possibility of such damages. Aureal reserves the right to make improvements in this manual and the products it describes at any time, without notice or obligation.

Copyright

Trademarks

Document Number: DO3010-030100

A3D Version 3.11

Contents

Chapter 1: Introduction to A3D

> *You are manning an anti-aircraft gun. You hear a jet approaching from your 6, flying low. You have no time to swivel, so you raise your gun and start firing as you hear the jet pass overhead. You hit it, and it explodes a moment later. You are ready to congratulate yourself, but have to swivel quickly to intercept a helicopter attacking from your 9.*

You wouldn't question the role of sound in this scenario for a minute in the real world. Positional sound is one of the main ways we orient ourselves in three dimensions. We know when airplanes are overhead before we look at them. The sounds tell us where to look. We manage this trick with only two ears.

You would emphatically question the scenario in a computer simulation or game running with normal stereo sound, or even “expanded” sound. Stereo sound just doesn't contain the cues you need to position sounds in space. On the other hand, programs written with A3D 3.0 accurately model positional audio, so that the scenario actually works. You really can shoot down incoming aircraft by ear. You can get this effect even with a pair of ordinary desktop speakers, and it gets even better with headphones or more than 2 speakers.

In this document, we'll explain the architecture of A3D, discuss how it works, how the different components fit together, and provide a reference manual page for every function. First, though, some background.

What is A3D?

A3D is a positional sound API and engine. It uses an engineering model of sound interacting with your ears and the environment that is based on Head Related Transfer Functions (HRTFs). Aureal and NASA helped pioneer the use of HRTFs. Psychoacoustics researchers all over the world have validated HRTFs as the most accurate way to recreate real-world audio with speakers or headphones.

Our brains can locate sounds in three dimensions (3D) because we have two separated ears with different frequency responses in different directions. This is also known as binaural hearing. Sounds located to the left reach the left ear before the right ear, and are louder in the left ear. Sounds located in front of us are brighter and louder than sounds behind us because our ears point slightly forward. Sounds located above us are distinguishable from sounds located below us — each ear is highly asymmetric. We also track sounds and resolve ambiguities by moving our head, and detect motion by hearing differences in loudness and frequency over time.

A3D 3.0 models the three-dimensional sound environment using HRTFs. In addition, it models room and environmental acoustics in real time. A3D 3.0 also follows sound reflections as they bounce around the environment. The noise of a fountain heard through a wall, and the noise of the fountain bounced off a wall, are different from the noise of a fountain heard directly.

What is the Goal of A3D?

The goal of the A3D 3.0 API is to make it as easy as possible for software developers to produce compelling, realistic audio for your title. Using as many or as few of the features of this SDK as you need, you can be sure that the A3D 3.0 engine provides you and your customers with the best possible audio experience regardless of the sound card used. To that end, it enables the developer to describe a world in an intuitive way and extracts relevant acoustic information from it. You don't need a Ph.D. in acoustics and you don't need to know the mathematics of the HRTFs to produce positional sound with A3D. We strived to make the concept of Wavetracing easy to understand, especially for 3D graphics programmers. Sound sources in A3D typically correspond to visual objects in 3D space; A3D takes care of calculating the effects of distance, echoes, absorption, and relative motion to determine the correct binaural sound to present to the listener.

An additional goal of A3D is to provide a consistent cross-platform experience. If Aureal Vortex audio hardware (or other A3D enabled hardware) is present at runtime, the A3D API will use it. A2D emulation lets A3D work on any sound card, even if Vortex hardware is not present and

A2D is used to augment the number of 3D channels available on any sound card. In addition, the A3D 3.0 engine automatically supports Microsoft's DirectSound3D (DS3D), Creative's EAX, and the IASIG's I3DL2 if present, for 3D sound and reverb functionality.

Positional 3D Audio

As we've already mentioned, we are able to locate sounds three dimensionally in the real world through a combination of time lag (also known as phase) and frequency variation — in other words, because our ears our separated in space and shaped asymmetrically. Once we locate what we hear, our ears tell our eyes where to look to see what's happening. Synchronized sound and visuals tell our brains that we are detecting real objects.

To produce the illusion of sonic virtual reality, A3D uses HRTF technology to re-create on the computer what our two ears would hear in the real world. This creates the illusion that we are detecting real objects. The illusion is heightened because the A3D technology is totally interactive. As we move or turn and change the direction we are facing, the sound coming to our ears changes appropriately, at the same time as the view on our screen changes.

There are obvious benefits to this approach: heightened realism, the feeling of immersion in the virtual reality, suspension of disbelief, increased interactivity, faster reaction times, and knowing where things are even if they are off screen. To accomplish all this, A3D needs you to define the position and orientation of each sound source, which can vary with time. In addition, you must define the position and orientation of one listener per frame, which again can vary with time. Given this information, the A3D engine dynamically calculates how to render the positional 3D audio scene and conveys it to the listener through the speakers or headphones.

3D Room Acoustics

By itself, direct path positional 3D audio does a good job of fooling our ear into locating a sound. However, the environment we are in greatly affects how sound is transmitted, and ultimately, how it is perceived. Sound doesn't always travel directly from the source to the listener; however, the environment affects how sound travels. Sound bounces off walls and ceilings, is absorbed by rugs, is scattered by hard objects, and finds its way to some degree under doors, around corners, and out windows. A drumbeat in an open field sounds much different from a drumbeat in a room because of the wall and ceiling echoes and reverberation in the room. The spacing of the echoes gives our ears cues about the size and shape of a room.

A3D 3.0 technology is extremely powerful and is capable of modeling the effects that environments have on sounds. To reproduce these effects, A3D uses Aureal Wavetracing™ technology. Wavetracing calculates how sound waves interact with the 3D environment: how they bounce off walls, pass through walls, and come around corners. Wavetracing is the sonic equivalent to ray tracing, which is used to produce photorealistic images.

Wavetracing provides even more realism, immersion and interactivity than simple direct path positional 3D audio. In fact, it is a necessary step towards true audio realism. A realistic audio environment creates a mood: a small, echoing space like a dungeon has a much different mood from a large, echoing space like a cathedral or concert hall, and both are quite different from an acoustically dead space.

With Wavetracing in place, sounds behind walls or around corners can tell you about objects that cannot be seen. So, for instance, in a role-playing game, the player might be able to hear the muffled sounds of monsters or characters moving on the other side of a closed door. That sort of cue can make all the difference in the game experience.

To calculate the acoustical effects of the 3D environment, A3D's Wavetracing engine needs you to define the position, shape and acoustic material type of the polygonal elements you want it to model. These elements might be based loosely on the visual elements of your simulated space, but shouldn't have as much detail. Wavetracing requires CPU cycles and since only relatively large objects affect sound, you can keep computational overhead low by only sending large, acoustically significant polygons to the Wavetracing engine.

The A3D API and Engine

A3D describes its world by default in terms of a right-handed Cartesian coordinate system, just like OpenGL. In this coordinate system (see Figure 1), the positive x-axis points to the right, the positive y-axis points up, and the positive z-axis points toward you.

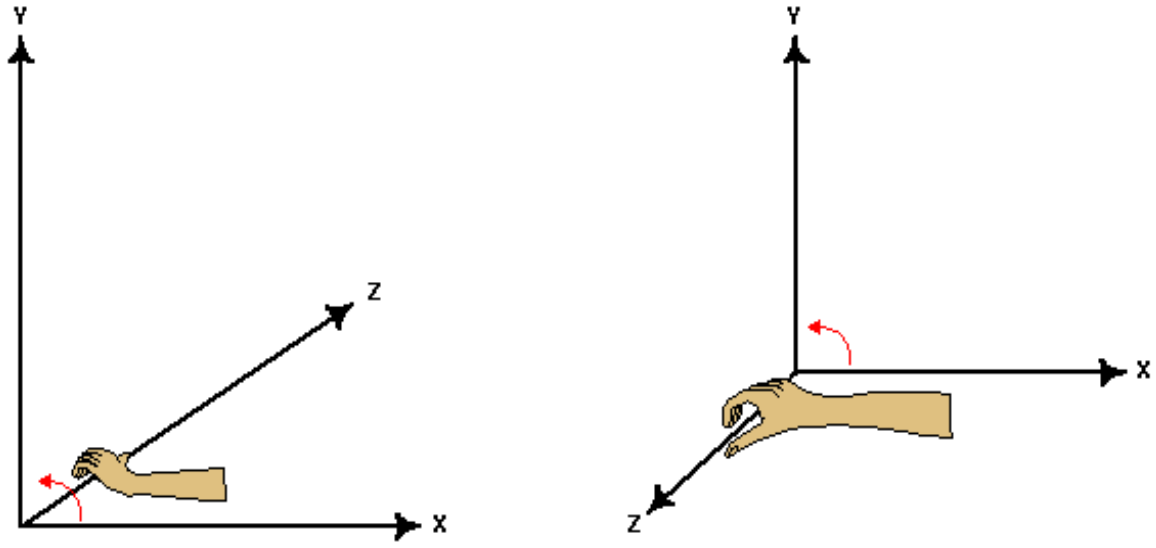


FIGURE 1. Co-ordinate Systems

The system is right-handed in the sense that your right thumb would point in the direction of the positive z-axis if you curled your fingers around the axis. Some 3D graphics systems, like Direct3D, use a left-handed coordinate system (shown at the left in Figure 1).

All objects in the A3D world are positioned in the space of its coordinate system. Just as a 3D graphics system only has one viewpoint at a time, the A3D audio system only has one listener at a time, with a specific position, orientation, and velocity. There may be many sound sources, each of which has a position, orientation, velocity, and volume cone as well as a wave data stream.

The A3D rendering engine can use Aureal Vortex hardware for maximum rendering fidelity and minimum impact on the system CPU. It can also run in software to complement limited hardware resources, or emulate hardware on non-accelerated systems. The rendering engine automatically scales across different platforms to produce the best sound each system can produce.

When the A3D engine is faced with a more complicated sound scenario than the rendering resources of the current system can support, it practices resource management — the software equivalent of triage. The A3D geometry engine computes the full audio scenario supplied by the application for the currently supported feature set, then passes the resulting frame buffer to the A3D resource manager. The resource manager ranks the sources in the buffer by loudness,

distance from the listener and application-specified priority, and sends them to the audio hardware (possibly via Aureal's A2D emulator or Microsoft's DirectSound3D), until it runs out of resources.

The last stage of the A3D pipeline takes care of any necessary filtering to produce the correct binaural sound for the current playback system — headphones, 2 speakers, 4 speakers, or whatever the user is using at the moment. The application doesn't have to worry about adapting to the speaker or sound card hardware at all. It only has to present its sources, listener, environment, materials, and geometry to the engine for rendering. The rest — geometry processing, psychoacoustics binaural rendering, hardware resource management, software fallback, output device specific filtering — is all done automatically by the A3D engine.

Summary

In this chapter, we have given you a very brief overview of A3D 3.0, positional 3D audio, and room acoustics. In the next chapter, we will discuss the A3D 3.0 architecture in more depth.

Chapter 2: A3D 3.0 Architecture Overview

As we mentioned in Chapter 1, A3D 3.0 is a positional sound API and engine. In this chapter, we'll discuss how an application presents a model of an acoustic space to the A3D API, and how the A3D engine turns that into positional audio.

A3D 3.0 is implemented as a COM[1] server. The developer accesses the functionality of the A3D 3.0 server through one or more defined interfaces that are defined according to the COM model. An application starts using the A3D engine by creating an instance of the A3dApi COM class with **CoCreateInstance** and calling the returned IA3d5 interface's Init method. The application then calls the IA3d5 interface's **QueryInterface** method to locate additional interfaces and calls their methods as needed to define a listener, sound sources, materials, and the scene's geometry. The A3D engine accumulates this data in a frame buffer. When the current scene is fully defined, the application calls the A3dApi object's Flush method to perform the Wavetracing calculations and send the audio scene from the process buffer to the resource manager and on to the audio hardware. The process is very similar to the way an application defines visual geometry for a 3D graphics engine like OpenGL or Direct3D.

[1]For more information about COM, we recommend Inside COM from Microsoft Press (ISBN 1-57231-349-8).

A3D Data Path

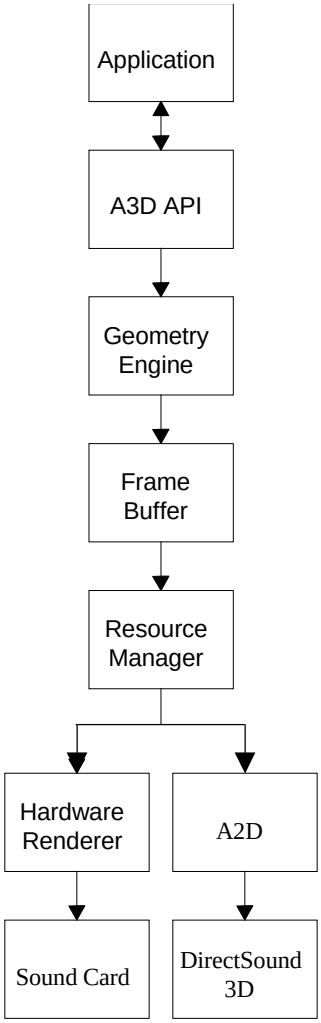


Figure2. A3D Data

Figure 2 illustrates the overall A3D data path.

The application calls the A3D API library through its COM interfaces to define the current audio scene. The API library passes the information to the geometry engine, which accumulates its output — a set of hardware and software controls for each source — in a buffer. When the engine sees the `Flush` method call, it performs the Wavetracing calculations and then sends the current audio frame to the resource manager. Typically the application sends one audio frame for each video frame.

The resource manager orders the sources in the buffer by their probable importance, taking into account audibility (calculated from distance and loudness), priority, and a weighting function specifying which is more important. The resource manager then processes each source in order. The resource manager either renders the audio stream using A2D, which is a software emulator, or sends it to the A3D system library that passes it on to the A3D sound card driver for rendering in Vortex hardware, or utilizes native DS3D hardware on non-Vortex hardware.

If the sound stream is rendered in hardware, a digital signal processor (DSP) on the sound card applies filtering and mixes to create binaural audio. Even if the sound stream is rendered via the A2D emulator, there is still a possibility of a hardware assist with the filtering and mixing if the sound card supports DirectSound3D. In any case, the resource manager continues to send sources to the rendering pipeline until it runs out of hardware voices. The exact number of hardware voices available is sound-card dependent. Please refer to the *A3D 3.0 Platform and Resource Manager Guide* for an overview of what is available on various hardware platforms.

Multi-channel Capability and Native Data Formats

A3D2 .0 is a great sound engine and API for 3D positional audio. It's also a great API and engine for other formats such as plain old stereo. A3D 3.0 enables a game to utilize mono and stereo content to be played back (rendered) in one of several ways:

- As true A3D sources.

  Stereo files get mono mixed.

- As "mono" sources.

  Where "mono" means that the 3D algorithm is turned off, but the data path (and the hardware resources consumed) are the same as for a true A3D source.

- As "native."

  Where "native" means that it's played back according to the format defined in the wave file and gets controls that correspond to that format. Thus, a mono file played natively gets L/R pan controls, as does a stereo file.

A3D 3.0 allows the defined "render" mode to be switched in real-time on any sound source that is under the care of the A3D 3.0 Resource Manager.

The table below shows which controls are available for which render modes.

Control	**Mono**	**Stereo**	**3D**
Gain	**X**	**X**	**X**
Pitch	**X**	**X**	**X**
Priority	**X**	**X**	**X**
Pan Control	**X**	**X**	
EQ (global t eble)			**X**
Position			**X**
Orientation			**X**
Cone			**X**
Distance			**X**
Doppler			**X**
Velocity			**X**
Geometry			**X**

The exact details for how you control this feature is in the API details. However, we can give you a sneak preview of what you're going to see. **IA3d5::NewSource**, a function you've already seen, plays an important role. It enables you to specify the initial render mode of the source. You can then use **IA3dSource2::Set/GetRenderMode** to your heart's content. But be careful! Switching modes while the source is playing is not strictly guaranteed to be click-free. Here are the functions that you'll use to control this feature:

```
IA3d5::NewSource(DWORD dwInitMode, IA3dSource2 **pSource);
IA3dSource2::SetRenderMode(DWORD dwRenderModeMask);
IA3dSource2::GetRenderMode(DWORD *pdwRenderModeMask);
IA3dSource2::SetPanValues(DWORD nChannels, LPA3DVAL fGains);
IA3dSource2::GetPanValues(DWORD nChannels, LPA3DVAL fGains);
```

IA3dSource2:Set/GetPanValues can be called at any time, but only takes effect if the source is in “native” mode. Only two-channel support is provided in this release. You must use the resource manager to use these features. Here's exactly what you can do:

- Mono as mono, played as 3D
- Stereo as mono, played as 3D
- Mono as stereo, played as 2 channel pan
- Stereo as stereo, played as 2 channel pan

No support for:

- Stereo as mono, played as 2 channel pan
- Anything with more than 2 channels
- Switching the mode of non-resource managed sources

The A3D Interface Hierarchy

Applications define acoustic geometry for A3D by using its COM interfaces. The top level or root interface for the A3D API is `IA3d5`. See Table 1 for a list of interface methods.

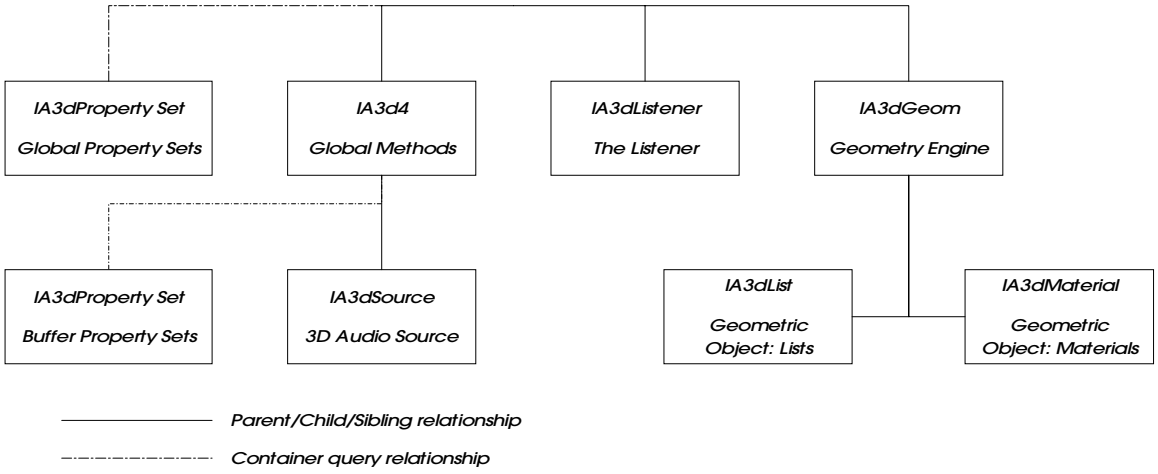


FIGURE 3. A3D COM Object Hierarchy

The root `IA3d5` interface can return pointers to its two peer interfaces `IA3dListener` and `IA3dGeom2` through its `QueryInterface` method. It can create an `IA3dSource2` interface using its `NewSource` or `DuplicateSource` methods.

Table 1. IA3d5 Interface Methods

IA3d5	**Standard COM IUnknown**	**A3D**	
Interface Methods	AddRef Release QueryInterface	A3dEnumerate BindReverb Clear Compat DisableViewer DuplicateSource Flush Init SetCooperativeLevel SetCoordinateSystem SetDistanceModelScale SetDopplerScale SetEq SetHFAbsorbFactor SetMaxHardwareSources SetMaxReflectionDelayTime SetNumFallbackSources SetOutputGain SetOutputMode SetResourceManagerMode SetRMPriorityBias SetStreamingProperties SetUnitsPerMeter	InitEx IsFeatureAvailable NewReverb NewSource RegisterApp RegisterVersion Shutdown UnlockFallbackAC3Decoder GetCooperativeLevel GetCoordinateSystem GetDistanceModelScale GetDopplerScale GetEq GetHardwareCaps GetHFAbsorbFactor GetMaxReflectionDelayTime GetNumFallbackSources GetOutputGain GetOutputMode GetResourceManagerMode GetRMPriorityBias GetSoftwareCaps GetStreamingProperties GetUnitsPerMeter

`IA3dPropertySet` defines the current property set. See Table 2 for a list of interface methods.

Table 2. IA3dPropertySet Interface Methods

IA3dPropertySet	**Standard COM IUnknown**	**Listener Property**	
Interface Methods	AddRef Release QueryInterface	AddInitalStateParameters QuerySupport Set	 Get

`IA3dListener` defines the current listener. See Table 3 for a list of interface methods.

Table 3. IA3dListener Interface Methods

IA3dListener	**Standard COM IUnknown**	**Listener Property**	
Interface Methods	AddRef Release QueryInterface	SetOrientation SetOrientationAngles SetPosition SetVelocity	GetOrientation GetOrientationAngles GetPosition GetVelocity

`IA3dReverb` defines the reverb properties. See Table 4 for a list of interface methods.

Table 4. IA3dReverb Interface Methods

IA3dListener	**Standard COM IUnknown**	**Listener Property**	
Interface Methods	AddRef Release QueryInterface	SetAllProperties SetPresetDamping SetPresetDecayTime SetPresetVolume SetReverbPreset	GetAllProperties GetPresetDamping GetPresetDecayTime GetPresetVolume GetReverbPreset

`IA3dSource2` defines a source of sound. See Table 5 for a list of interface methods.

Table 5. IA3dSource2 Interface Methods

IA3dSource2	**Standard COM IUnknown**	**A3D**	
Interface Methods	AddRef Release QueryInterface	AllocateAudioData ClearPlayEvents FreeAudioData LoadWaveData LoadFile Lock SetAudioFormat SetCone SetDistanceModelScale SetDopplerScale SetEq SetGain SetMinMaxDistance	Play Rewind Rewind Stop Unlock GetAudibility GetAudioFormat GetCone GetDistanceModelScale GetDopplerScale GetEq GetGain GetMinMaxDistance GetOcclusionFactor

IA3dSource2	Standard COM IUnknown	A3D	
		SetOrientation	GetOrientation
		SetPanValues	GetPanValues
		SetPitch	GetPitch
		SetPlayPosition	GetPlayPosition
		SetPlayTime	GetPlayTime
		SetPosition	GetPosition
		SetPriority	GetPriority
		SetReflectionDelayScale	GetReflectionDelayScale
		SetReflectionGainScale	GetReflectionGainScale
		SetRenderMode	GetRenderMode
		SetReverbMix	GetReverbMic
			GetStatus
		SetTransformMode	GetTransformMode
			GetType
		SetVelocity	GetVelocity
		SetVolumetricBound	GetVolumetricBounds
		SetVolumetricDamping	GetVolumetricDamping

`IA3dGeom2` is the interface for the A3D geometry engine, which deals with raw 3D transformation matrices, materials, and quadrilateral elements. See Table 6 for a list of interface methods.

Table 6. IA3dGeom2 Interface Methods

IA3dGeom2	Standard COM IUnknown	Low-level A3D Geometry	
Interface	AddRef	Begin	NewList
Methods	Release	BindEnvironment	NewMaterial
	QueryInterface	BindListener	Normal
		BindMaterial	PopMatrix
		BindSource	PushMatrix
		Disable	Rotate
		Enable	Scale
		End	Tag
		IsEnabled	Translate
		LoadIdentity	Vertex
		LoadMatrix	
		MultMatrix	
		NewEnvironment	
			GetMatrix
		SetOcclusionMode	GetOcclusionMode
		SetOcclusionMode	GetOcclusionMode
		UpdateInterval	UpdateInterval

IA3d eom2	Standard COM IUnknown	Low-level A3D Geometry	
		SetOpeningFactor SetPolygonBloatFac or SetReflectionDelay cale SetReflectionGainS ale SetReflectionMode SetReflectionUpdat e- Interval SetRenderMode	 GetPolygonBloatF ctor GetReflectionDelc Scale GetReflectionGair Scale GetReflectionMoc e GetReflectionUpd te- Interval GetRenderMode

You may find some of the geometry methods familiar. The matrix, translation, rotation, vertex, and normal methods correspond exactly to similarly named geometric transformation functions in OpenGL. `IA3dGeom2` can create materials (`IA3dMaterial`) and data lists (`IA3dList`) with its `NewMaterial` and `NewList` methods. The various `Bind` methods set the state for their objects to the current transformation matrix stack. The `Begin` and `End` methods of `IA3dGeom2` bracket the vertices and normals for the primitive specified in the `Begin` call. To define geometry, you typically set up the transformation matrix stack, call `Begin`, define the vertices and normals, and call `End`.

`IA3dMaterial` defines acoustic materials. See Table 7 for a list of methods.

Table 7. IA3dMaterial Interface Methods

IA3dl laterial	Standard COM IUnknown	A3D	
Inter ace Metl ods	AddRef Release QueryInterfac	Duplicate Load Save SetNameID SetReflectance SetTransmittance	SelectPreset Serialize Unserialize GetClosestPreset GetNameID GetReflectance GetTransmittance

`Load` and `Save` read and write disk files. `UnSerialize` and `Serialize` read and write packed memory images.

`IA3dList` is the interface for lists of geometry and state data. See Table 8 for a list of methods.

Table 8. IA3dList Interface Methods

IA3dList	Standard COM IUnknown	A3D
Interface Methods	AddRef Release QueryInterface	Begin Call End EnableBoundingVol

Lists are used define objects that might be added to the geometry more than once. `Begin` and `End` bracket the list definition, and `Call` inserts the list into the geometry.

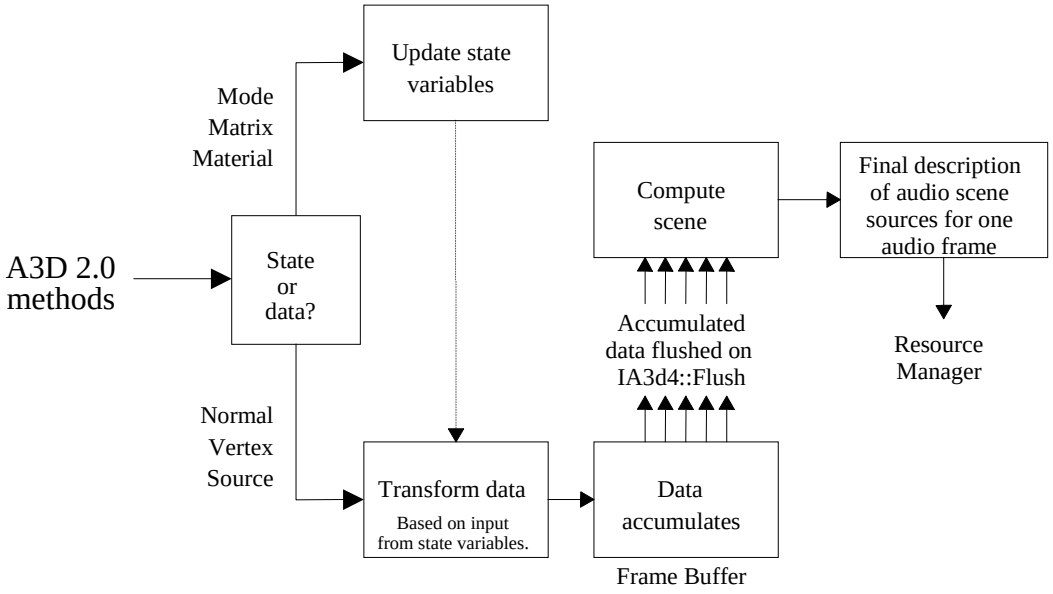


FIGURE 4. A3D API COM Method Data Flow Diagram

Figure 4 illustrates what happens inside the A3D API engine when an application calls an A3D API COM method. In Windows, the engine is implemented as a dynamic-link library, a3dapi.dll, which must be installed in either the same directory as your application or the Windows system directory. As shown in the diagram above, each call to an A3D 3.0 API method updates the current audio scene in some way (just as it did under A3D 2.0). It might change a state, add a source, position the listener, add polygons or, in the case of **IA3d5::Flush**, trigger the final scene description to be computed and sent to the resource manager. The resource manager then passes the audio frame to the audio hardware or to the rendering software, depending on the system configuration.

Data

In A3D, geometric information is data, defined by the vertices and normals of points, lines, and quadrilaterals. Data defines the position, orientation and velocity of sources and listeners, and the position and orientation of walls and openings.

States

States, on the other hand, modify the way the geometric data behaves in the audio scene. There are three kinds of states:

- Acoustic material
- Transformation matrix
- Rendering mode

Acoustic Material

Every acoustic material has reflectance and occlusion factors for both low and high frequencies. A cork wall occludes almost all the high frequency sound and most of the low frequency sound that reaches it, reflecting very little of either. A metal wall exhibits much higher reflectance and lower occlusion. The materials properties are states that are explicitly bound to geometric data with a method call.

Transformation Matrix

The transformation matrix is another kind of state. Throughout the process of defining a scene, the application applies translations and rotations to the frame of reference. When geometric vertex data is defined, the current transformation matrix in effect is applied to the data before it is added to the frame buffer. When a source is bound to the scene (with `IA3dGeom2::BindSource`), the current transformation matrix in effect is applied to the source's position data.

Rendering Mode

Rendering modes constitute a third kind of state. For instance, the application can globally enable or disable first order reflection calculations on hardware that supports this feature. With reflections enabled, the user might get echoes; without reflections, there might be more resources available for other acoustic Wavetracing calculations.

Wavetracing Algorithms

When the A3D engine renders a scene, it applies the available (depending on the platform) enabled rendering modes. The rendering modes are, in order of increasing complexity: direct path, occlusion, and first order reflections.

Direct Path

The engine determines the path of the sound emitted from each source along a straight line through the environmental media to the listener. This rendering mode is always enabled and available on all platforms.

The sound (audio file input) coming from the source is modified the by the following:

- Source's cone parameters and orientation
- Sound's gain and equalization
- Source's velocity relative to the listener and Doppler shift factor

The sound is attenuated based on the following:

- Distance between source and listener
- Distance model scale factor and cutoff distances of the source
- Attenuation rate and high-frequency absorption rate of the environment

Occlusion

The engine determines the path of the sound emitted from each source and follows it along a straight line through the environmental media and any intervening polygons or walls to the listener. Occlusion comes into play when the source and listener are on the opposite sides of a wall. This rendering mode can be disabled. It is available on all platforms.

The sound is attenuated based on the following:

- Distance between source and listener
- Source and environment factors
- Low- and high-frequency occlusion factors of the intervening materials

First Order Reflections

The engine does all the same calculations as it would for occlusion, but in addition it takes into account reflections from any intervening polygons or walls, based on the low- and high-frequency reflection factors of the intervening materials. Reflection comes into play when the source and listener are on the same side of a wall and is added to the sound from the direct path calculation. First order reflections can be disabled. It is not available on all platforms.

Polygon Complexity

As we've seen, acoustic Wavetracing must follow the sound from each source to the listener, taking into account the possible occlusion or reflection of each polygon in the audio scene. The different rendering modes have varying costs, but the overall CPU usage of the Wavetracing calculation is still directly proportional to the product of the number of sources in the scene and the number of polygons.

$$(\text{Wavetracing CPU usage}) \propto (\text{Number of Source} \times \text{Number of Polygons})$$

If an application tries to use highly detailed video geometry to define the audio scene, computer performance may be hampered, as the A3D engine attempts to render tens of thousands of quadrilaterals. Fortunately, an audio scene doesn't need such detail.

Visible light has wavelengths measured in microns, but that isn't the limit to object visibility. Our eyes can resolve objects smaller than a millimeter close up, and objects of a few centimeters across a room. Audible sound, on the other hand, has wavelengths measured in centimeters (for high frequencies) to meters (for low frequencies). We can resolve the echoes and occlusions from objects roughly the size of our ears if they are close to our ears, and from objects several meters in size if they are across a room.

This means that audio scenes can be described with dramatically fewer polygons than video scenes. Doing so at the level of raw geometry requires you to simplify or reduce the number of polygons in a video scene.

To summarize, represent a given audio scene with a very rough polygonal representation. Extra detail doesn't necessarily improve the user's experience.

Summary

A3D is a positional sound API and engine. An application calls the A3D engine through the engine's interfaces. Each scene has one listener position and as many audio sources and polygons as needed. Far fewer audio polygons are needed to define a scene than video polygons. The number of audio polygons should be kept low to conserve CPU time. The A3D engine performs Wavetracing calculations on completed audio scenes and passes the results to a resource manager, which sends the most important source information to the audio hardware.

Chapter 3: Functional Summary

This chapter gives a brief listing and description of each method under a particular interface. The methods are further described in Chapter 4: “A3D Direct Path Reference Pages”, Chapter 5: “Property Set Reference Pages”, and Chapter 6: “Geometry Engine Reference Pages”. The interfaces are presented in the following order:

- IA3d5
- IA3dPropertySet
- IA3dListener
- IA3dSource2
- IA3dPropertySet
- IA3dGeom2
- IA3dList
- IA3dMaterial

IA3d5 Interface Methods

IA3d5	
Method Name	**Description**
A3dEnumerate	Enumerates all audio output devices.
AddRef	Increments the **IA3d5** reference count.
BindReverb	Binds an A3D reverb preset to the world.
Clear	Clears all data for an audio frame.
Compat	Sets a compatibility mode.
DisableViewer	Disables the A3D shared memory interface.
DuplicateSource	Duplicates an audio source.
Flush	Flushes all data for an audio frame.
GetHardwareCaps	Query the hardware for 3D and stereo support.
GetSoftwareCaps	Query the software (A2D) engine for 3D and stereo support.
Init	Initializes the A3D root object.
InitEx	Initializes the A3D root object
IsFeatureAvailable	Checks if a requested feature is available in the hardware.
NewReverb	Creates a new reverb object.
NewSource	Creates a new source with no data.
QueryInterface	Returns an interface pointer for a supported interface.
RegisterApp	Unsupported.
RegisterVersion	Tells the library what version the application was built with.
Release	Decrements the **IA3d5** reference count.
Set/GetCooperativeLevel	Sets and gets the audio device cooperative level.
Set/GetCoordinateSystem	Sets and gets the coordinate system for geometry data.
Set/GetDistanceModelScale	Globally scales the distance model attenuation curve.
Set/GetDopplerScale	Globally scales the effect of Doppler.
Set/GetEq	Sets and gets the global equalization for all sources.
Set/GetHFAbsorbFactor	Unsupported.
Set/GetMaxReflectionDelayTiı e	Sets and gets the maximum delay possible for reflections.
Set/GetNumFallbackSources	Sets and gets the number of fallback software channels.
Set/GetOutputGain	Sets and gets the global output gain for all A3D sources.
Set/GetOutputMode	Sets and gets the speaker output mode.

IA3d5	
Set/GetResourceManagerMode	Unsupported.
Set/GetRMPriorityBias	Sets the weight of priority to audibility for all resource managed sources.
Set/GetStreamingProperties	Modifies control values used by the streaming engine.
Set/GetUnitsPerMeter	Specifies the number of application units in a meter.
SetMaxHardwareResources	Sets the maximum number of hardware sources that can be allocated by the resource manager.
Shutdown	Releases all A3D interfaces and any resources associated with them.
UnlockFallbackAC3Decoder	Checks for valid key to enable Dolby Digital AC-3 fallback decoder.

IA3dListener Interface Methods

IA3dListener Interface	
Method Name	**Description**
AddRef	Increments the **IA3dListener** reference count.
QueryInterface	Returns an interface pointer for a supported interface.
Release	Decrements the **IA3dListener** reference count.
Set/GetOrientation	Sets and gets the orientation of the listener.
Set/GetOrientationAngle	Sets and gets the orientation of the listener.
Set/GetPosition	Sets and gets the position of the listener.
Set/GetVelocity	Sets and gets the velocity of the listener.

IA3dReverb Interface Methods

IA3dReverb Interface	
Method Name	**Description**
AddRef	Increments the **IA3dReverb** reference count.
QueryInterface	Returns an interface pointer for a supported interface.
Release	Decrements the **IA3dReverb** reference count.
Set/GetAllProperties	Sets or gets all properties of a custom reverb, or modifies ll preset properties at once.
Set/GetPresetDamping	Sets or gets the damping factor for this reverb object's active p set.
Set/GetPresetDecayTime	Sets or gets the decay time for this reverb object's active preset
Set/GetPresetVolume	Sets or gets the volume for this reverb object's active preset.
Set/GetReverbPreset	Selects a reverb preset for this reverb object, or returns th current preset.

IA3dSource2 Interface Methods

IA3dSource2 Interface	
Method Name	**Description**
AddRef	Increments the **IA3dSource2** reference count.
AllocateAudioData	Allocates memory for audio data.
ClearPlayEvents	Clears all audio events for a source.
FreeAudioData	Releases allocated data for a sound source.
GetAudibility	Gets the calculated audibility of the source.
GetOcclusionFactor	Gets the occlusion factor of the source.
GetStatus	Gets the activity status of a source.
GetType	Gets the type of the source.
GetAudioSize	Gets the size of the audio file in bytes.
LoadWaveData	Loads audio data from memory into the source.
LoadWaveFile	Loads data into the sound source from file.
Lock	Allows data to be written to a buffer.
Play	Starts a sound source playing.
QueryInterface	Returns an interface pointer for a supported interface.
Release	Decrements the **IA3dSource2** reference count.
Rewind	Rewinds a sound source back to the beginning of the wave data
Set/GetAudioFormat	Sets and gets the format of the audio information.
Set/GetCone	Sets the directionality of a source cone.
Set/GetDistanceModelScale	Changes the distance attenuation curve for a source.
Set/GetDopplerScale	Sets and gets the exaggerated Doppler effect on a source.
Set/GetEq	Sets the tonal equalization of a source.
Set/GetGain	Sets and gets the playback gain of the source.
Set/GetMinMaxDistance	Sets and gets the range over which the distance model will be applied to a source.
Set/GetOrientation	Sets and gets the direction of the sound source.
Set/GetOrientationAngles	Sets and gets the orientation of the sound source.
Set/GetPanValues	Sets the gains for multi-channel, non-spatialized sources.
Set/GetPitch	Set and gets the pitch bend of the source.
Set/GetPlayPosition	Sets the playback cursor in a sound source to a particular time.
Set/GetPlayTime	Sets and gets the playback cursor in the audio data.

IA3dSource2 Interface	
Set/GetPosition	Sets and gets the location of a sound source.
Set/GetPriority	Sets and gets the priority of a source.
Set/GetReflectionDelayScale	Scales the reflection delays for a source.
Set/GetReflectionGainScale	Scales the reflection gains for a source.
Set/GetRenderMode	Set and gets the render mode of a source.
Set/GetReverbMix	Sets or gets the mix level in the current reverb for this source.
Set/GetTransformMode	Sets and gets the transform mode for a source.
Set/GetVelocity	Sets and gets the velocity of a source.
Set/GetVolumetricBounds	Sets or gets the mix level in the current reverb for this source.
Set/GetVolumetricDamping	Sets and gets the damping factors used to rendering the volumetric source.
SetPlayEvent	Sets an event to be triggered at a certain point in the audio data
Stop	Stops a playing sound source.
Unlock	Unlocks a previously locked sound source.

IA3dPropertySet Interface Methods

IA3dListener Interface	
Method Name	**Description**
AddInitialStateParameters	Sets the initial “zero” state of a property set.
AddRef	Increments the **IA3dPropertySet** reference count.
Get	Retrieves data for an item in a property set.
QueryInterface	Returns an interface pointer for a supported interface.
QuerySupport	Determines whether a property in a property set is supported o the port or device.
Release	Decrements the **IA3dPropertySet** reference count.
Set	Sets data for an item in a property set.

IA3dGeom2 Interface Methods

IA3dGeom2 Interface	
Method Name	**Description**
AddRef	Increments the **IA3dGeom2** reference count.
Begin	Begins to insert vertex and normal data for a geometric primitive.
End	Finishes vertex data block.
BindEnvironment	Unsupported.
BindListener	Inserts the listener into the scene hierarchy.
BindMaterial	Sets the current material.
BindSource	Inserts a source into the scene hierarchy.
Disable	Globally disables a feature in the Wavetracing engine.
Enable	Globally enables a feature in the Wavetracing engine.
GetMatrix	Gets the current matrix on the stack.
IsEnabled	Returns whether or not a feature is enabled in the Wavetracing engine.
LoadIdentity	Loads an identity matrix onto the matrix stack.
LoadMatrix	Loads an arbitrary matrix onto the matrix stack.
MultMatrix	Multiplies the current matrix by an arbitrary matrix.
NewEnvironment	Unsupported.
NewList	Creates a new list of geometry data.
NewMaterial	Creates a new material.
Normal	Specifies a normal for a polygon.
PopMatrix	Pops a matrix off the matrix stack.
PushMatrix	Pushes a matrix onto the matrix stack.
QueryInterface	Returns an interface pointer for a supported interface.
Release	Decrements the **IA3dGeom2** reference count.
Rotate	Applies a rotational transformation to the current matrix.
Scale	Applies a scale transformation to the current matrix.
Set/GetOcclusionMode	Unsupported.
Sets/GetOcclusionUpdateInterval	Sets the number of frames between occlusion processing.
SetOpeningFactor	Sets the opening factor for subfaces.
Set/GetPolygonBloatFactor	Sets the reflection bloat factor for polygons.
Set/GetReflectionDelayScale	Sets the delay scaling factor for reflections.

IA3dGeom2 Interface	
Set/GetReflectionGainScale	Sets the gain scaling factor for reflections.
Set/GetReflectionMode	Unsupported.
Set/GetReflectionUpdateInterv l	Sets the number of frames between reflection processing.
Set/GetRenderMode	Sets the current render mode for polygon processing.
Tag	Tags the next polygon.
Translate	Applies a translation to the current matrix.
Vertex	Send vertex data for a primitive to the rendering engine.

IA3dList Interface Methods

IA3dList Interface	
Method Name	**Description**
AddRef	Increments the **IA3dList** reference count.
Begin	Begins adding data to a list.
End	Closes the begin block.
Call	Executes the sequence of commands stored in a list object.
EnableBoundingVol	Enables bounding box culling for a list.
QueryInterface	Returns an interface pointer for a supported interface.
Release	Decrements the **IA3dList** reference count.

IA3dMaterial Interface Methods

IA3dMaterial Interface	
Method Name	**Description**
AddRef	Increments the **IA3dMaterial** reference count.
Duplicate	Unsupported.
GetClosestPreset	Unsupported.
Load	Unsupported.
QueryInterface	Returns an interface pointer for a supported interface.
Release	Decrements the **IA3dMaterial** reference count.
Save	Unsupported.
SelectPreset	Unsupported.
Serialize	Unsupported.
Set/GetNameID	Sets and gets the name ID of the material.
Set/GetReflectance	Sets the reflectance of a material.
Set/GetTransmittance	Sets the transmittance of a material.
UnSerialize	Unsupported.

Chapter 4: A3D Direct Path Reference Pages

The basic technology in A3D 3.0 is called direct path rendering which is the term used to represent sound propagating on a direct path from the source of the sound to the listener's ears. It does not include secondary effects such as reflections — sound bouncing off of walls — or occlusions — sound being transmitted through walls. This chapter contains all the material pertinent to programming direct path rendering into your application. Full positional 3D audio is possible when programming at this level.

Direct path rendering is accessed through three interfaces:

- IA3d5
- IA3dListener
- IA3dSource2

IA3d5 is the top-level root interface to A3D. All other interfaces are obtained through it in either of the following ways:

- Querying for a pointer to an interface that **IA3d5** creates automatically.
- Using an **IA3d5** method to create an object then getting an instance of an interface for it returned.

Each of the following three sections gives a short introduction to each interface, followed by reference pages for the methods contained within it.

IA3d5 Interface

The root interface to A3D is **IA3d5**. This is the top-level interface and the one from which the other interfaces are either queried or created. A3D itself is started by getting a handle to this interface. The following code shows how to initialize A3D:

```
/* initialize COM */
CoInitialize(NULL);


/* load the a3dapi.dll and get a handle to IA3d5 */
hr = CoCreateInstance(CLSID_A3dApi, NULL, CLSCTX_INPROC_SERVER,
IID_IA3d5, (void **)&pA3d5);


/* if this failed then the dll wasn't on the system */
if (FAILED(hr))
return(hr);
```

At this point, the DLL is loaded and the **IA3d5** interface is available to the application. Next, initialize A3D with the features the application is going to use. Features are requested during initialization because some require hardware to be reserved. The initial state of requested features is off. The application can switch them on or off as necessary.

Typically, an A3D 3.0 application uses the following initialization settings:

```
hr = pA3d5->InitEx(NULL, A3D_OCCLUSIONS | A3D_1ST_REFLECTIONS,
A3DRENDERPREFS_DEFAULT);
```

Assuming the call succeeds, A3D is initialized and ready to accept instructions.

After basic initialization, the application can access the other methods contained within **IA3d5**.

To shut down, the application should release each interface and finally **IA3d5** itself:

```
/* release IA3d5 */
pA3d5->Release();

/* shut COM down */
CoUninitialize();
```

IA3d5::A3dEnumerate

Enumerates all audio output devices.

Prototype

HRESULT **A3dEnumerate(**
    LPA3DENUMCALLBACK *lpA3dEnumCallback*,
    LPVOID *lpContext*
);

Parameters

lpA3dEnumCallback	callback function
lpContext	context data value

Return Values

S_OK
E_POINTER
E_FAIL

Description

This function is used to determine which audio device on the system A3D 3.0 will render through. After the device has been identified by using **A3dEnumerate**, the id is passed into **IA3d5::Init**. From here on A3D will render through that device.

For each detected audio output device on the system, a single call is made to the A3DENUMCALLBACK function with relevant data. The result of this is that the calling routine will end up with an id specifying the audio device it wants to render with. The lpContext value is set by the calling application and passed in to each call to lpA3dEnumCallback unchanged.

See Also

IA3d5::InitEx

A3DENUMCALLBACK

IA3d5::AddRef

Increments the IA3d5 reference count.

Prototype

ULONG **AddRef**(
    void
);

Parameters

None.

Return Values

Returns the new reference count.

Description

When going through a COM method such as **QueryInterface** or **NewSource** to get an interface pointer to a component, the reference count of the component is automatically incremented. The reference count is used to let the component know when nothing is accessing it anymore and that it can delete itself from memory.

Whenever an interface pointer is assigned to another interface pointer, the **AddRef** method should be called to let the component know that two pointers are using the same interface. Now when the **Release** method is called, the component won't delete itself since it has been told something else is still using it. Consider the following example:

```
hr = pRoot->QueryInterface(IID_IBox, (void **)&pBox1);
if (SUCCEEDED(hr))
{
   pBox1->DrawIt();
   pBox2 = pBox1;
   pBox2->AddRef();
   pBox1->Release();
}
```

While *pBox1* is now invalid because it has been released, *pBox2* remains intact and can still be used.
All A3D 3.0 interfaces inherit the COM **IUnknown** interface, which contains the methods **AddRef**, **QueryInterface**, and **Release**. *Inside COM* by Microsoft Press is an excellent resource for detailed information on COM.

See Also

IA3d5::QueryInterface

IA3d5::Release

IA3d5::BindReverb

Binds an A3DReverb to the world.

Prototype

HRESULT **BindReverb**(
    LPA3DREVERB *pReverb*
);

Parameters

pReverb A pointer to the A3DReverb object to be bound.

Return Values

S_OK
E_INVALIDARG

Description

Enables a reverb by binding an A3DReverb object to the world. In the current implementation, enabling a reverb will disable any reverb that is currently enabled. This may change in the future as multiple reverbs may be allowed.

If NULL is passed in, all reverberation effects are disabled.

Note: All functions that set reverb properties can be used on any reverb object whether or not the reverb object is currently bound.

Note: Switching between reverbs will occur smoothly, at the next call to **IA3d5::Flush**..

See Also

IA3d5::NewReverb

IA3dReverb

IA3d5::Clear

Clears all data for an audio frame.

Prototype

```
HRESULT Clear(
    void
);
```

Parameters

None.

Return Values

S_OK

Description

Use **IA3d5::Clear** to remove the audio and geometry information for the next frame.

A frame of audio is similar to a frame of graphics; while the previous frame is playing, new data describing the next frame to be rendered is accumulated in a second frame buffer. The data in this second frame buffer is not applied to any sound sources until the entire scene has been described. When the scene is complete, the application signals that the data in the second frame buffer should be used instead of the original data. In graphics, this is known as *double buffering* and the concept for audio is the same. An audio frame in this sense is a collection of parameters describing how the scene should be rendered. The 'data' being referred to here is *not* the audio wave sample data, but rather the parameters describing how the wave data should be filtered.

An application delimits its audio frame with calls to **IA3d5::Clear** and **IA3d5::Flush**, and makes calls to other A3D 3.0 methods between them. It's necessary to call **IA3d5::Clear** at the beginning of the frame to remove the data that accumulated during the previous frame. **IA3d5::Clear** removes *only* data — all of the states are left intact. States include the current matrix, the matrices the listener or any sources are bound to, the current material, and any rendering modes. Keeping this information in mind, do not use **IA3d5::Clear** as a "reset" button.

See Also

IA3d5::Flush

IA3d5::Compat

Sets a compatibility mode.

Prototype

```
HRESULT Compat(
    DWORD    dwMode,
    DWORD    dwValue
);
```

Parameters

dwMode	Identifies a compatibility mode to set.
dwValue	Value to be sent.

Return Values

S_OK

Description

IA3d5::Compat allows an application to access undocumented features within A3D 3.0. Currently, there are no undocumented features so this method should not be called.

See Also

None.

IA3d5::DisableViewer

Disables the A3D shared memory interface.

Prototype

HRESULT **DisableViewer**(
    void
);

Parameters

None.

Return Values

S_OK

Description

Debug Viewer versions of the A3D 3.0 library support a shared memory interface, which among other things, is used to send information to the A3D GL Debug Viewer application. This viewer shows a wireframe view of the audio database, with the listener, sources and polygons displayed. Since the viewer displays wireframe it is possible to see through walls which means it could be used to cheat in multiplayer games if a user managed to get debug A3D binaries and a copy of the viewer. **IA3d5::DisableViewer** prevents this by disabling the shared memory interface.

If used at all, this method should only be incorporated in a game at the last moment before the game is shipped, as the debug viewer is an invaluable tool during development.

Once disabled, the shared memory can't be re-enabled until A3D has been shut down and re-initialized.

See the Chapter 3: “Debug Viewer GL” on page 13 of the *A3D 3.0 Users' Guide* for more information.

See Also

None.

IA3d5::DuplicateSource

Duplicates an audio source.

Prototype

HRESULT **DuplicateSource**(
    LPA3DSOURCE22 *pOriginal*,
    LPA3DSOURCE22 **ppCopy*
);

Parameters

pOriginal	Pointer to the source to be copied.
ppCopy	Address of a pointer to a new **IA3dSource2**. The function fills out the pointer value.

Return Values

S_OK
E_INVALIDARG
A3DERROR_NOT_VALID_SOURCE
A3DERROR_NO_WAVE_DATA
A3DERROR_FAILED_DUPLICATION

Description

IA3d5::DuplicateSource creates a copy of an existing source. This is similar to **IA3d5::NewSource** in that it returns a pointer to an instance of an **IA3dSource2** interface that represents a new sound source object. The duplicate is always of the same type as the original, and the wave data of the original and duplicate is shared, but all other properties can be modified independently.

After a source has been duplicated, it can be released without affecting the duplicate.

See Also

IA3d5::NewSource

IA3d5::Flush

Flushes all data to the rendering engine at the end of a frame.

Prototype

```
HRESULT Flush(
    void
);
```

Parameters

None.

Return Values

S_OK

Description

This method signals the end of an audio frame. It reads all data received since the last **IA3d5::Clear** call which describes the scene to be rendered, computes the parameters required to render that scene, and sends the parameters to the A3D resource manager. Everything that the application does before calling **IA3d5::Flush** is accumulated in a frame buffer but the effect of it only becomes audible when this method is called. For example, when a source is told to play, it doesn't play immediately but starts playing when **IA3d5::Flush** is next called. Deferring results of the instructions issued during a frame to the end of that frame allows synchronization of all sounds and reduces computation.

Note that an audio frame in this sense is only a collection of parameters describing how the scene should be rendered. The ‘data’ being referred to here is *not* the audio wave sample data, but rather the parameters describing how the wave data should be filtered. Without calling **IA3d5::Flush**, the audio wave data will continue to be fed to the sound card at the correct rate without any stalls or interruptions. This method is purely for updating the parameters that define the filters applied to the sources being played.

Depending on the A3D implementation, **IA3d5::Flush** may not return immediately as parts of the computation carried out by the geometry renderer may be in the same thread as the calling function.

See Also

IA3d5::Clear

IA3d5::GetHardwareCaps

Query the hardware for 3D and stereo support.

Prototype

HRESULT **GetHardwareCaps**(
    A3DCAPS_HARDWARE **pCaps*
);

Parameters

pCaps	The address of an A3DCAPS_HARDWARE structure created by the application.

Return Values

S_OK
E_POINTER
A3DERROR_FUNCTION_NOT_VALID_BEFORE_INIT

Description

This method is used to find out some basic information about the hardware in the system. The return value fills out an A3DCAPS_HARDWARE structure that contains the following fields:

```
typedef struct __A3DCAPS_HARDWARE
{
      DWORD dwSize;
      DWORD dwFlags;
      DWORD dwReserved;
      DWORD dwReserved2;
      DWORD dwOutputChannels;
      DWORD dwMinSampleRate;
      DWORD dwMaxSampleRate;
      DWORD dwMax2DBuffers;
      DWORD dwMax3DBuffers;
} A3DCAPS_HARDWARE;
```

dwFlags specifies the type of hardware and is either A3D_DIRECT_PATH_A3D or A3D_DIRECT_PATH_GENERIC where the latter means it's not an A3D card.

dwOutputChannels is always 2.

dwMinSampleRate and *dwMaxSampleRate* are the minimum and maximum sample rates that are supported, and for most Vortex-based devices will be 0 and 96,000 respectively. This doesn't mean that a 96 kHz wave file can be loaded, but rather that a 48 kHz wave file can be played back with a pitch shift factor of 2.

dwMax2DBuffers and *dwMax3DBuffers* report the number of stereo and 3D buffers the hardware can support. These numbers assume that either creating a 3D source doesn't take resources away from stereo sources, or that there are no stereo sources, and vice versa. For example, a card might report it can do 23 stereo sources and 8 3D sources, but in reality if you create 8 3D sources it will not allow even a single stereo source. The other caveat is that the wave files are of an optimal format, usually 22 kHz 8 bit mono.

The numbers reported in this structure are best case and assume nothing else is attempting to use the hardware.

See Also

IA3d5::GetSoftwareCaps.

IA3d5::GetSoftwareCaps

Query the software (A2D) engine for 3D and stereo support.

Prototype

HRESULT **GetSoftwareCaps**(
    A3DCAPS_SOFTWARE **pCaps*
);

Parameters

pCaps The address of an A3DCAPS_SOFTWARE structure created by the application.

Return Values

S_OK
E_POINTER
A3DERROR_FUNCTION_NOT_VALID_BEFORE_INIT

Description

This method is used to find out some basic information about the A2D engine. The return value fills out an A3DCAPS_SOFTWARE structure that contains the following fields:

```
typedef struct __A3DCAPS_SOFTWARE
{
      DWORD dwSize;
      DWORD dwVersion;
      DWORD dwFlags;
      DWORD dwReserved;
      DWORD dwReserved2;
      DWORD dwOutputChannels;
      DWORD dwMinSampleRate;
      DWORD dwMaxSampleRate;
      DWORD dwMax2DBuffers;
      DWORD dwMax3DBuffers;
} A3DCAPS_HARDWARE;
```

Currently only dwMinSampleRate, dwMaxSampleRate, dwMax2DBuffers, and dwMax3DBuffers contain useful information.

dwMinSampleRate and *dwMaxSampleRate* are the minimum and maximum sample rates that are supported, and will be around 0 and 96,000 respectively. This doesn't mean that

a 96 kHz wave file can be loaded, but rather that a 48 kHz wave file can be played back with a pitch shift factor of 2.

dwMax2DBuffers and *dwMax3DBuffers* report the number of stereo and 3D buffers the engine can support. These numbers assume that either creating a 3D source doesn't take resources away from stereo sources, or that there are no stereo sources, and vice versa.

See Also

IA3d5::GetHardwareCaps.

IA3d5::Init

Initializes the A3D 3.0 audio library.

Prototype

```
HRESULT Init(
    LPGUID    lpGuid,
    DWORD     dwFeatures,
    DWORD     dwRenderPrefs
);
```

Parameters

lpGuid — Pointer to the GUID from the **DeviceEnumeration** function.

dwFeatures — Specifies the rendering features required by the application. It is a bitwise OR of any of the following values:

A3D_1ST_REFLECTIONS

A3D_DISABLE_FOCUS_MUTE

A3D_DISABLE_SPLASHSCREEN

A3D_GEOMETRIC_REVERB

A3D_OCCLUSIONS

A3D_REVERB

dwRenderPrefs — Reserved flag for future rendering engine options. Specify:

A3DRENDERPREFS_DEFAULT

Return Values

S_OK
A3DERROR_FAILED_INIT

Description

It is recommended that **IA3d5::InitEx** be used in place of **IA3d5::Init**.

A3D 3.0 must be initialized before an application can play audio. Achieve this by calling the **IA3d5::Init** method. Nearly all A3D 3.0 methods fail if they are called before **IA3d5::Init**.

IA3d5::Init has three arguments. The first, *lpGuid*, is an ID for the audio device the application wants to use to render A3D. This is passed as an LPGUID and it uniquely

identifies a particular audio device. If the application wants to use the system default audio device, this argument should be NULL.

Some rendering features, such as reflections, require hardware to be reserved, so these features must be requested at initialization. The features required by the application are passed in the second argument, *dwFeatures*, as a bitmask of the values listed above. If a particular feature is not available, the call still succeeds and A3D still runs but without that feature. Use **IA3d5::IsFeatureAvailable** to find out if a request for a feature is successful. Requesting a feature doesn't enable it — hardware is merely reserved for the feature. As long as a feature is available, it can be turned on or off at any time after initialization.

The third argument, *dwRenderPrefs*, should always be A3DRENDERPREFS_DEFAULT.

Note that normally, the A3D 3.0-enabled application must have the focus in order to render audio. If the app loses focus, the output streams are muted. A3D 3.0 introduces DISABLE_FOCUS_MUTE as a new value that will prevent the streams from being muted when focus is lost, thus allowing apps that run in the background to still produce audio.

See Also

IA3d5::IsFeatureAvailable
IA3dGeom2::Enable
IA3dGeom2::Disable
IA3d5::Set/GetNumFallbackSources
IA3d5::Release

IA3d5::InitEx

Initializes the A3D root object.

Prototype

HRESULT **InitEx(**
    LPGUID *lpGuid*,
    DWORD *dwFeatures*,
    DWORD *dwRenderPrefs*,
    HWND *hWnd*,
    DWORD *dwCoopLevel*
);

Parameters

lpGuid — Pointer to the GUID from the **DeviceEnumeration** function.

dwFeatures — Specifies the rendering features required by the application. It is a bitwise OR of any of the following values:

A3D_1ST_REFLECTIONS

A3D_DISABLE_FOCUS_MUTE

A3D_DISABLE_SPLASHSCREEN

A3D_GEOMETRIC_REVERB

A3D_OCCLUSIONS

A3D_REVERB

dwRenderPrefs — Reserved flag for future rendering engine options. Specify:

A3DRENDERPREFS_DEFAULT

hWnd — in, handle for Resource manager

dwCoopLevel — in, cooperative level

Return Values

S_OK
A3DERROR_FAILED_INIT

Description

A3D 3.0 must be initialized before an application can play audio. Achieve this by calling the **IA3d5::InitEx** method. Nearly all A3D 3.0 methods fail if they are called before

IA3d5::InitEx. This function replaces the old initialization method of calling **IA3d4::Init** and then having to call **IA3d4::SetCooperativeLevel**.

IA3d5::InitEx has three arguments. The first, *lpGuid*, is an ID for the audio device the application wants to use to render A3D. This is passed as an LPGUID and it uniquely identifies a particular audio device. If the application wants to use the system default audio device, this argument should be NULL.

Some rendering features, such as reflections, require hardware to be reserved, so these features must be requested at initialization. The features required by the application are passed in the second argument, *dwFeatures*, as a bitmask of the values listed above. If a particular feature is not available, the call still succeeds and A3D still runs but without that feature. Use **IA3d5::IsFeatureAvailable** to find out if a request for a feature is successful. Requesting a feature doesn't enable it — hardware is merely reserved for the feature. (Therefore, it is advised not to As long as a feature is available, it can be turned on or off at any time after initialization.

The third argument, *dwRenderPrefs*, should always be A3DRENDERPREFS_DEFAULT.

Note that normally, the A3D 3.0-enabled application must have the focus in order to render audio. If the app loses focus, the output streams are muted. A3D 3.0 introduces DISABLE_FOCUS_MUTE as a new value that will prevent the streams from being muted when focus is lost, thus allowing apps that run in the background to still produce audio.

See Also

IA3d5::IsFeatureAvailable
IA3dGeom2::Enable
IA3dGeom2::Disable
IA3d5::Set/GetNumFallbackSources
IA3d5::Release

IA3d5::IsFeatureAvailable

Checks the features available to the application after initialization.

Prototype

```
HRESULT IsFeatureAvailable(
    DWORD    dwFeature
);
```

Parameters

dwFeature	Specifies the feature to query. It should be one of the following (*not* a bitmask): A3D_1ST_REFLECTIONS A3D_OCCLUSIONS

Return Values

TRUE	Feature is available.
FALSE	Feature not available.

Description

Features are requested in a parameter passed in **IA3d5::InitEx** but the parameter is exactly that — a request. It doesn't mean all the features are available. To determine what features are available to the application after initialization, use the method **IA3d5::IsFeatureAvailable**. It returns TRUE if the feature was requested and is available and FALSE if the feature wasn't requested or isn't available.

See Also

IA3d5::InitEx

IA3d5::NewReverb

Creates a new reverb object.

Prototype

HRESULT **NewReverb**(
    LPA3DREVERB **ppReverb*
);

Parameters

ppReverb The address of a pointer to an IA3dReverb. The function fills out the pointer.

Return Values

S_OK
A3DERROR_FEATURE_NOT_REQUESTED
A3DERROR_FEATURE_NOT_SUPPORTED
A3DERROR_MEMORY_ALLOCATION

Description

Creates a new reverb and allocates the memory for the data structure. The reverb is initially set to the A3DREVERB_PRESET_GENERIC preset.

In order to create an A3DReverb object, reverb must have been requested in the **IA3d5::Init** or **IA3d5::InitEx** call by including the A3D_REVERB flag. If this was not the case, A3DERROR_FEATURE_NOT_REQUESTED will be returned.

If reverb is not available, A3DERROR_FEATURE_NOT_SUPPORTED is returned.

To determine if reverb is going to run in hardware or software, use **IA3d5::GetHardwareCaps**.

Note: Though the preferred way of freeing A3DReverb objects (and all A3D objects) is to call the COM Release method, the **IA3d5::Shutdown** function will release all A3DReverb objects created.

See Also

IA3d5::BindReverb

IA3dReverb

IA3d5::NewSource

Creates a new source.

Prototype

```
HRESULT NewSource(
    DWORD           dwFlags,
    LPA3DSOURCE22   *ppSource
);
```

Parameters

dwFlags Specifies the properties and initial state of the source. It is a bitwise OR of the following:

A3DSOURCE_TYPEDEFAULT

A3DSOURCE_INITIAL_RENDERMODE_A3D

A3DSOURCE_INITIAL_RENDERMODE_NATIVE

A3DSOURCE_TYPEUNMANAGED

A3DSOURCE_TYPESTREAMED

ppSource The address of a pointer to an IA3dSource2. The function fills out the pointer value.

Return Values

S_OK
E_INVALIDARG
A3DERROR_MEMORY_ALLOCATION
A3DERROR_FAILED_CREATE_PRIMARY_BUFFER

Description

IA3d5::NewSource creates a new audio source and allocates the memory for the data structure. The value of *dwFlags* determines the type of source being created and its initial render mode. If *dwFlags* is set to A3DSOURCE_TYPEDEFAULT the source will be a resource managed A3D source that is set up for static (as opposed to streaming) wave data.

A 3D sources can be positioned in 3D space and are affected by any geometry and atmospheric properties. Typically, any sound that moves around or has a specific position in the world should be an A3D source. Native sources can be panned left and

right, are not affected by geometry or the atmosphere, and are useful for playing back music or sound effects that are pre-encoded with spatial information. The two initial render modes are exclusive - if both are set the source will be created as a native source.

While the render mode chosen here is the initial render mode, it can't be changed later for unmanaged sources. Only managed sources can be freely switched between native and A3D modes while they are playing. See the A3D *3.0 Platform Guide and Resource Manager* document for information on managed and unmanaged sources.

Use A3DSOURCE_TYPESTREAMED if you intend to dynamically stream wave data into the source.

No memory or wave data is allocated here, only memory for the data fields in the source. As such, it's rare that this method fails. Wave data memory is allocated when the application calls either **IA3dSource2::LoadWaveFile** or **IA3dSource2::AllocateAudioData**. Check the return codes of these methods.

A source must be released when it is no longer needed. This frees any memory and resources allocated to it, and allows them to be used by another source. Use **IA3dSource2::Release** for this purpose.

See Also

IA3dSource2::LoadFile
IA3dSource2::AllocateAudioData
IA3dSource2::Release
IA3dSource2::SetRenderMode

IA3d5::QueryInterface

Returns an interface pointer for a supported interface.

Prototype

```
HRESULT QueryInterface(
    REFIID          iid,
    LPVOID FAR      *pInterface
);
```

Parameters

iid	Interface identifier. Specify one of: IID_IA3dListener IID_IA3dGeom2 IID_IA3d5
pInterface	Address of a pointer to an interface which will be filled out by the method.

Return Values

S_OK
E_NOINTERFACE

A3DERROR_FAILED_INIT_QUERIED_INTERFACE

Description

All A3D interfaces inherit the **IUnknown** interface that contains a method called **Query-Interface**. This method is used to let the application know what other interfaces a particular interface supports, and to return a pointer to a requested interface if it is supported. The different A3D interfaces support different interfaces.

From **IA3d5::QueryInterface**, the following interfaces are available: IID_IA3dListener, which will return a pointer to the **IA3dListener** interface, IID_IA3dGeom2, which will return a pointer to the **IA3dGeom2** interface, and IID_IA3d5, itself, which will return another pointer to the **IA3d5** interface and increment its reference count.

Calling any **QueryInterface** and asking for an interface that isn't supported will return the error E_NOINTERFACE. The address of the pointer passed in to the method will be left at the value it was set to by the calling method, so it may not be NULL. For this reason, it is essential to check the return value of this method.

If a valid interface is requested in the call to **IA3d5::QueryInterface**, but something has gone wrong during the A3D initialization, the method will return A3DERROR_FAILED_INIT_QUERIED_INTERFACE.

See Also

IA3d5::AddRef
IA3d5::Release

IA3d5::RegisterApp

Unsupported.

Prototype

HRESULT **RegisterApp**(
    REFIID *riid*
);

Parameters

riid

Return Values

S_OK

Description

None.

See Also

None.

IA3d5::RegisterVersion

Tell the library what version the application was built with.

Prototype

```
HRESULT RegisterVersion(
    DWORD dwVersion
);
```

Parameters

dwVersion	Specifies the version of A3D the app was compiled with. Should always be A3D_CURRENT_VERSION.

Return Values

S_OK
A3DERROR_FUNCTION_NOT_VALID_BEFORE_INIT

Description

This method is used to stamp an application with a record of the A3D version it was compiled with. Currently the call is ignored but future releases may use this to enable older algorithms for applications that depended on a particular behavior.

See Also

None.

IA3d5::Release

Decrements the IA3d5 reference count.

Prototype

```
ULONG Release(
    void
);
```

Parameters

None.

Return Values

Returns the new reference count.

Description

When going through a COM method such as **QueryInterface** or **NewSource** to get an interface pointer to a component, the reference count of the component is automatically incremented. The reference count is used to let the component know when nothing is accessing it anymore and that it can delete itself from memory.

Calling **IA3d5::Release** decrements the reference count for the **IA3d5** interface, and if it is 0, the object deletes itself from memory.

Note that **IA3d5**, **IA3dGeom2**, and **IA3dListener** all share the same reference count, as they are simply different interfaces onto the same base object. Only when all three have been released will the reference count of any one of them be 0.

All A3D 3.0 interfaces inherit the COM **IUnknown** interface that contains the methods **AddRef**, **QueryInterface**, and **Release**. *Inside COM* by Microsoft Press is an excellent resource for detailed information on COM.

See Also

IA3d5::AddRef
IA3d5::QueryInterface

IA3d5::Set/GetCooperativeLevel

Sets and gets the cooperative level for the application.

Prototype

HRESULT **SetCooperativeLevel**(
    HWND   *hWnd*,
    DWORD   *dwLevel*
);

HRESULT **GetCooperativeLevel**(
    DWORD   **pdwLevel*
);

Parameters

hWnd	Window handle of the application.
dwLevel	Specify A3D_CL_NORMAL or A3D_CL_EXCLUSIVE for the priority.

Return Values

S_OK
E_INVALIDARG
A3DERROR_FAILED_SETCOOPERATIVE_LEVEL

Description

Several applications can run on a single audio device at the same time. Use **IA3d5::SetCooperativeLevel** immediately after **IA3d5::Init** to determine the level of access an application has to the audio device.

The first argument specifies the window handle of the application. This is necessary so that A3D can keep track of when the window loses input focus.

When *dwLevel* is set to A3D_CL_NORMAL (the recommended setting), the audio device being used is left available to other applications. However, the level of access to the hardware is dependent on the particular implementation of A3D. For example, on Vortex2 systems, only one application may have access to the hardware required for rendering reflections at any time, whether reflections are actually being rendered or not. To stop any other applications from running on the same audio device, *dwLevel* should be set to A3D_CL_EXCLUSIVE.
Use of this method is mandatory — it has to be called because it's the only way A3D can reliably get the window handle of the application.

See Also

IA3d5::InitEx

IA3d5::Init

IA3d5::Set/GetCoordinateSystem

Sets and gets the coordinate system.

Prototype

```
HRESULT SetCoordinateSystem(
    DWORD    dwCoordSystem
);

HRESULT GetCoordinateSystem(
    DWORD    *pdwCoordSystem
);
```

Parameters

dwCoordSystem	Specifies the coordinate system to use. Can be either A3D_RIGHT_HANDED_CS or A3D_LEFT_HANDED_CS.

Return Values

S_OK

Description

There are two systems in Euclidean geometry for specifying coordinates: right-handed and left-handed systems. In a right-handed system, positive X goes right, positive Y goes up, and positive Z comes out of the screen *toward* you. A left-handed system has Z going the other way, positive into the screen *away* from you. Most graphics systems and virtually all graphics textbooks use a right-handed coordinate system, and this is the default for A3D 3.0. However, some use a left-handed system (in fact, A3D 1.0 uses a left-handed system), so the **IA3d5::SetCoordinateSystem** method lets an application select the system that matches the graphics library being used.

If an application wants to call **IA3d5::SetCoordinateSystem**, it should call it immediately after initialization. Typically, it should be called after **IA3d5::InitEx** (or **IA3d5::Init** and **IA3d5::SetCooperativeLevel**). Once the coordinate system is set, it can never be changed.

The selected coordinate system is applied to all 3D data that is sent to A3D, whether that data is source position, listener orientation, or the coordinates of a vertex, for example.

The choice is there purely for convenience but there may be a very slight performance gain when using a right-handed system, as this is the system native to A3D 3.0.

See Also

IA3d5::InitEx

IA3d5::Set/GetCooperativeLevel

IA3d5::Set/GetDistanceModelScale

Globally scales the distance model attenuation curve.

Prototype

HRESULT **SetDistanceModelScale**(
    A3DVAL *fScale*
);

HRESULT **GetDistanceModelScale**(
    A3DVAL **fScale*
);

Parameters

fScale	Non-negative floating point number specifying the scale factor applied to the distance model.

Return Values

S_OK

E_INVALIDARG

Description

IA3d5::SetDistanceModelScale globally changes the attenuation rate of sources due to distance from the listener. *fScale* changes the slope of the attenuation curve after a source is beyond its minimum distance.

The default value is 1.0, which means the gain of a source will be reduced by 6 dB for each doubling in distance, starting at the minimum distance. Values less than 1.0 reduce the effect of distance by stretching the curve out, and values greater than 1.0 increase the effect.

This scale factor is applied in addition to any distance model scale factors specified for each particular source. In common with all global/local scalars, *fScale* is multiplied by the source factor, which is set using **IA3dSource2::SetDistanceModelScale**.

See IA3dSource2::SetMinMaxDistance and IA3dSource2::SetDistanceModelScale for information on the distance model.

See Also

IA3dSource2::Set/GetDistanceModelScale

IA3dSource2::Set/GetMinMaxDistance

IA3d5::Set/GetDopplerScale

Globally scales the effect of Doppler.

Prototype

HRESULT **SetDopplerScale**(
    A3DVAL *fScale*
);

HRESULT **GetDopplerScale**(
    A3DVAL **fScale*
);

Parameters

fScale Non-negative floating point number specifying the Doppler multiplier.

Return Values

S_OK

E_INVALIDARG

Desription

The Doppler effect is the change in pitch of a sound caused by the motion of the listener and the object making the sound through air. Sounds traveling towards a listener appear to have a higher pitch, and those traveling away have a lower pitch.
IA3d5::SetDopplerScale globally applies a scale factor to the Doppler effect for all sources. *fScale* is used to change the effective speed of sound for Doppler calculations, thereby exaggerating or diminishing the effect.

If 0.0 < *fScale* < 1.0, the speed of sound is increased, reducing the amount of pitch bend for any given object speed. If *fScale* > 1.0, the speed of sound is reduced, increasing the amount of pitch bend. 0.0 turns Doppler shifting completely off, and 1.0 (the default) leaves the speed of sound unchanged at 340m/s.

This scale factor is applied in addition to any doppler scale factor specified for each particular source. In common with all global/local scalars, *fScale* is multiplied by the source factor, which is set using **IA3dSource2::SetDopplerScale**.

See Also

IA3dSource2::Set/GetDopplerScale

IA3dSource2::Set/GetPitch

IA3dSource2::Set/GetVelocity

IA3d5::Set/GetEq

Sets the global equalization for all sources.

Prototype

```
HRESULT SetEq(
    A3DVAL fEq
);

HRESULT GetEq(
    A3DVAL *fEq
);
```

Parameters

fEq Floating point number between 0.0 and 1.0 inclusive.

Return Values

S_OK

E_INVALIDARG

Description

IA3d5::SetEq globally applies an equalization effect to all sources. It is similar in effect to a treble control on a stereo system and is completely independent of distance and gain. It is low-pass only and doesn't allow high frequencies to be boosted.

If 0.0 < *fEq* < 1.0, high frequencies are attenuated more as *fEq* approaches 0.0. The default setting of 1.0 means there is no additional high frequency attenuation applied to sources.

This method is useful for simulating different environments. For example, *fEq* = 0.3 would make everything sound like it was underwater.

This EQ value is applied in addition to any EQ value specified for each particular source. In common with all global/local scalars, *fEq* is multiplied by the source EQ value, which is set using **IA3dSource2::SetEq**.

See Also

IA3dSource2::Set/GetEq

IA3d5::Set/GetHFAbsorbFactor

Unsupported

Prototype

HRESULT **SetHFAbsorbFactor**(
    FLOAT *fFactor*
);

HRESULT **GetHFAbsorbFactor**(
    FLOAT **fFactor*
);

Parameters

fFactor

Return Values

A3DERROR_UNIMPLEMENTED_FUNCTION

Description

This was an old method used to set attenuate the high frequencies for all sources. Use **IA3d5::SetEq** instead.

See Also

IA3d5::Set/GetEq

IA3d5::Set/GetMaxReflectionDelayTime

Sets the maximum delay possible for reflections.

Prototype

HRESULT **SetMaxReflectionDelayTime**(
    A3DVAL *fSeconds*
);

HRESULT **GetMaxReflectionDelayTime**(
    A3DVAL **fSeconds*
);

Parameters

fSeconds Maximum delay time in seconds between a reflection and its direct path.

Return Values

S_OK
E_INVALIDARG

Description

Rendering reflections requires that some amount of the direct path audio stream be kept around after it has been played so that a delayed version of it can be played back later. The longer the delay between the direct path and the last reflection, the greater the amount of data that has to be stored.

IA3d5::SetMaxReflectionDelayTime allows an application to specify the maximum time difference between the direct path and the longest reflection. The default of 0.3 seconds is adequate for modeling spaces up to the size of a football stadium. Reflections with delays greater than the time set by this method are clamped to the maximum delay but attenuation due to distance is computed normally.

This method is not a geometry method, which is why it is in **IA3d5** instead of **IA3dGeom2** with the other reflection methods. It is used to allocate memory in the driver, and this memory is allocated as long as reflections are successfully requested in the call to **IA3d5::InitEx**, whether they are actually enabled and used or not. The time set by **IA3d5::SetMaxReflectionDelayTime** is not scaled by **IA3dGeom2::SetReflectionDelayScale**.

Negative values for *fSeconds* cause this method to return E_INVALIDARG.

See Also

IA3d5::InitEx

IA3d5::Set/GetNumFallbackSources

Sets the number of fallback software channels.

Prototype

HRESULT **SetNumFallbackSources**(
    DWORD *dwNumSources*
);

HRESULT **GetNumFallbackSources**(
    DWORD **dwNumSources*
);

Parameters

dwNumSources Number of fallback sources to be allocated.

Return Values

S_OK
E_FAIL
A3DERROR_FUNCTION_NOT_VALID_BEFORE_INIT

Description

This method enables you to specify the number of A2D sources. If you don't specify a number of A2D sources the default value of 12 is used. In cases where it is necessary to play more sources concurrently than the hardware is able to handle, A2D is able to play the less important sources in software. Importance is a function of source priority and audibility, with the bias between those two properties being specified in a call to **IA3d5::SetRMPriorityBias**.

This method can't be called before A3D is initialized by **IA3d5::InitEx**, but the number of fallback sources can be changed dynamically.

In the absence of A3D hardware, A2D can completely fill in, using its internal renderer to play sources in software or making use of other 3D hardware on the system. A2D is the backup audio engine for A3D and emulates much of the functionality of A3D in software.

See Also

IA3d5::Set/GetRMPriorityBias

IA3dSource2::Set/GetPriority

IA3d5::Set/GetOutputGain

Sets and gets the global output gain for all A3D sources.

Prototype

HRESULT **SetOutputGain**(
    A3DVAL *fGain*
);

HRESULT **GetOutputGain**(
    A3DVAL **fGain*
);

Parameters

fGain Global output gain.

Return Values

S_OK
E_INVALIDARG

Description

IA3d5::SetOutputGain is the master volume control for all A3D, A2D and DS3D sources. Changing the output gain globally and uniformly scales the gains of all sources and reflections.

fGain is in the range 0.0 to 1.0, where 0.0 is silence and 1.0 is 0 dB. Each reduction by half represents a 6 dB attenuation. This method is the global equivalent of **IA3dSource2::SetGain**.

IA3d5::GetOutputGain gets the current global output gain setting.

See Also

IA3dSource2::SetGain

IA3d5::Set/GetOutputMode

Sets and gets the speaker output mode.

Prototype

HRESULT **SetOutputMode**(
    DWORD *dwXTalk1*,
    DWORD *dwXTalk2*,
    DWORD *dwMode*
);

HRESULT **GetOutputMode**(
    DWORD **dwXTalk1*,
    DWORD **dwXTalk2*,
    DWORD **dwMode*
);

Parameters

dwXTalk1 Specifies the cross-talk mode for output 1. Specify one of OUTPUT_HEADPHONES, OUTPUT_SPEAKERS_WIDE, OUTPUT_SPEAKERS_NARROW.

dwXTalk2 Specifies the cross-talk mode for output 2. Specify one of OUTPUT_HEADPHONES, OUTPUT_SPEAKERS_WIDE, OUTPUT_SPEAKERS_NARROW.

dwMode Specifies whether two stereo outputs are used for quad or stereo. Specify either OUTPUT_MODE_STEREO or OUTPUT_MODE_QUAD.

Return Values

S_OK
E_INVALIDARG
A3DERROR_FUNCTION_NOT_VALID_BEFORE_INIT

Description

The output mode of the audio device determines what type of cross-talk algorithm, if any, to apply to the final signal. A cross-talk canceller is used when listening to speakers to eliminate the signal from the left speaker reaching the right ear and vice versa.

There are three output modes available. OUTPUT_MODE_HEADPHONES disables the cross-talk canceller since, when wearing headphones, each ear hears only the near speaker. OUTPUT_MODE_NARROW enables the cross-talk canceller with an

algorithm designed for speakers placed quite close together, as they are with speakers built into a monitor. OUTPUT_MODE_WIDE enables a different algorithm, which is designed for speakers placed further apart, for example when a pair of satellite speakers are placed on opposite sides of the monitor. Some A3D devices support two outputs. Whether those can be set to different output modes or not depends on the particular device.

The *dwMode* parameter is used to switch A3D devices which support two sets of speakers between outputting a pair of binaural signals and a quad-speaker output where the first output feeds the front speakers and the second the rear speakers.

See Also

None.

IA3d5::Set/GetResourceManagerMode

Unsupported

Prototype

HRESULT **SetResourceManagerMode**(
    DWORD *dwMode*
);

HRESULT **GetResourceManagerMode**(
    DWORD **dwMode*
);

Parameters

dwMode Resource Manager mode flags.

Return Values

A3DERROR_UNIMPLEMENTED_FUNCTION

Description

This is an old method that should not be used. The resource management mode is specified per source in **IA3d5::NewSource**.

See Also

IA3d5::NewSource

IA3dSource2::GetType

IA3d5::Set/GetRMPriorityBias

Sets the weight of priority to audibility for all resource-managed sources.

Prototype

HRESULT **SetRMPriorityBias**(
    A3DVAL *fBias*
);

HRESULT **GetRMPriorityBias**(
    A3DVAL **fBias*
);

Parameters

fBias A number in the range 0.0 to 1.0 for priority bias.

Return Values

S_OK
E_INVALIDARG

Description

The resource manager determines whether to play a source or not based on a weight it computes for that source. This weight is a function of audibility and priority. Audibility is calculated internally by A3D and takes into account source attenuation due to distance, gain, equalization and occlusions. As such, audibility is not directly controlled by the application. Priority is set by the application and is used to determine how important a source is.

While the priority of a source can be set by an application, setting the priority to maximum may still not be enough to guarantee that the source is played if its audibility is low. The method **IA3d5::SetRMPriorityBias** is used to bias the weight calculation towards priority, allowing the resource manager algorithm to place more importance on either priority or audibility.

The default value is 0.5, which places equal weight on priority and audibility. Values greater than 0.5 bias the calculation towards priority, and values less than 0.5 bias it towards audibility. The resource manager algorithm is *weight* = (*fAud**(1 - *fBias*)) + (*fPriority* * *fBias*), where *fBias* is provided by this method, *fPriority* by **IA3dSource2::SetPriority**, and *fAud* is computed internally by A3D.

See Also

IA3d5::NewSource

IA3dSource2::Set/GetPriority

IA3d5::Set/GetStreamingProperties

Modifies control values used by the streaming engine.

Prototype

HRESULT **SetStreamingProperties**(
    DWORD *dwBufferLength*
    DWORD *dwThreadPriority*
);

HRESULT **GetStreamingProperties**(
    DWORD **dwBufferLength*
    DWORD **dwThreadPriority*
);

Parameters

dwBufferLength The amount of data to buffer, in milliseconds.

dwThreadPriority The priority of the streaming thread. The accepted values are:

A3D_STREAMING_PRIORITY_NORMAL (default)

A3D_STREAMING_PRIORITY_HIGH

A3D_STREAMING_PRIORITY_HIGHEST

Return Values

S_OK
A3DERROR_STREAMING_BUFFER_LENGTH
A3DERROR_STREAMING_PRIORITY

Description

The amount of data to buffer is specified in milliseconds, so as to be independent of the wave data format. This number can be directly converted from a time value into a byte value, and is the minimum amount of data that should be "queued" up to play at any given time. The default value is 900 ms (0.9s) - three times the maximum reflection delay time, 0.3s. Setting the buffer length will *only* affect subsequent calls to **IA3dSource2::LoadFile**. Existing streams will remain unchanged. When the buffer is larger than the file being played, the buffer will automatically be resized to the size of the file.

If the application uses a lot of high priority threads, or is very CPU intensive, there may be dropouts in the streaming, If this happens, use this parameter to raise the thread's priority.

Note: The safest time to set the thread priority is before any streaming sources start playing. However, making this call does not immediately change the streaming thread's priority. It is determined internally when the thread can change priority.

A3DERROR_BUFFER_LENGTH will be returned if the value requested for dwBufferLength greater than 10,000 (ms).

A3DERROR_STREAMING_PRIORITY will be returned if the value requested for dwThreadPriority is invalid.

See Also

None.

IA3d5::Set/GetUnitsPerMeter

Specifies the number of application units in a meter.

Prototype

```
HRESULT SetUnitsPerMeter(
    A3DVAL fUnits
);

HRESULT GetUnitsPerMeter(
    A3DVAL *fUnits
);
```

Parameters

fUnits	Floating point number specifying the number of units in a meter.

Return Values

S_OK

E_INVALIDARG

Description

This method is used to tell the A3D library what units the application is using. By default, A3D expects everything in meters - positions and vectors are measured in meters and velocities in meters per second - so the default value is 1.0. Specifying another value applies a conversion inside A3D to compensate for the different units.

For example, if an application is specifying everything in kilometers, it would set *fUnits* to 0.001f. If it is using inches, *fUnits* should be set to 39.37f.

The effect of calling this method is not retroactive. Data already sent to A3D, whether it is the location of a source or a list of polygons cached in a list, is not modified. It only affects future data. For this reason, it is inadvisable to use **IA3d5::SetUnitsPerMeter** dynamically. It should be set once at the beginning before any sources are created and left at that value.

Irrespective of what the current units are, the default minimum and maximum distance values for a source are 1m and 5000m respectively. This means that calling **IA3dSource2::GetMinMaxDistance** before explicitly setting the minimum and maximum distances will return different values depending on the current units since it is returning 1m and 5000m converted to application units. For example, if *fUnits* is 10, calling **IA3dSource2::GetMinMaxDistance** will return 10 and 50000 as the default values. If *fUnits* is 0.2, the default values returned will be 0.2 and 1000.

There is no performance impact from using units other than meters.

See Also

IA3dSource2::Set/GetMinMaxDistance

IA3d5::SetMaxHardwareSources

Sets the maximum number of hardware sources that can be allocated by the resource manager.

Prototype

HRESULT **SetMaxHardwareSources(**
    DWORD *nMaxHardwareFallback*
);

Parameters

nMaxHardwareFallback maximum number of hardware sources to use

Return Values

S_OK
E_INVALIDARG
A3DERROR_FUNCTION_NOT_VALID_AFTER_INIT

Description

The A3D resource manager, by default, will use as many hardware resources as it needs. If a machine has a sound card with 16 available hardware streams and the application attempts to play 20, all 16 hardware resources will be used. However, some applications need to occasionally lock down a hardware source (such as for streaming a large file from disk). If an application knows ahead of time that it will need a hardware resource it can use **IA3d5::GetHardwareCaps** to determine how many total hardware resources are available, decide how many resources it needs to reserve, and then pass the difference into **IA3d5::SetMaxHardwareSources**.

This function only applies to the resource manager. Unmanaged sources can be allocated only if they're available. Also this function can only be allocated before **IA3d5::InitEx** is called. Otherwise the error code A3DERROR_FUNCTION_NOT_VALID_AFTER_INIT is returned.

While not documented, there is an **IA3d5::GetMaxHardwareSources** function call that allows you to see how many sources you have previously allocated, should you need that information.

See Also

IA3d5::InitEx

IA3d5::GetHardwareCaps

IA3d5::Shutdown

Releases all A3D interfaces and any resources associated with them.

Prototype

HRESULT **Shutdown**(
    void
);

Parameters

None.

Return Values

S_OK

Description

IA3d5::Shutdown is a convenience method which blindly releases all A3D interfaces and the memory associated with them, removing the need to call the **Release** method for each object the application has created. It also ensures that any other A3D resources, such as hardware audio channels, are properly shut down.

Some care should be exercised when using this method. **IA3d5::Shutdown** can't null out any pointers to interfaces the application has, so the following code would cause a crash:

```
pIA3dGeom2->NewList(&pDungeon);
pIA3d5->Shutdown();
pDungeon->Begin();
```

After calling **IA3d5::Shutdown**, any A3D interface pointers are invalid and the application should set them to NULL. The same care has to be taken when using any of the interface **Release** methods, but in that case it is easy to keep track of the invalid interface pointers since they are manually released one at a time.

See Also

IA3d5::Release

IA3d5::UnlockFallbackAC3Decoder

Checks for valid key to enable Dolby Digital AC-3 support. Dolby Digital software fallback will only work if it is unlocked. A game must set a valid key to A3D, before using Dolby Digital software fallback decoding.

Prototype

HRESULT **UnlockFallbackAC3Decoder(**
    LPSTR *szKey*,
    DWORD *dwReserved*
);

Parameters

szKey	Key for unlocking Software decoding AC-3
dwReserved	Don't use

Return Values

S_OK
A3DERROR_INVALID_AC3_KEY

Description

In order for an application to use Dolby Digital decoding, a licensing agreement has to be in place with Dolby. In most cases, getting the AC-3 content requires such a license, and receiving A3D support is simple. Contact devsupport@a3d.com for more information about receiving a Dolby Digital license.

Once the license is in place, a unique key will be generated for this application. The key takes the form of a text string. Pass it in to this function once at initialization time and AC-3 decoding will be available for this A3D session. Once the object is destroyed, UnlockFallbackAC3Decoder must be called again.

See Also

None.

A3DENUMCALLBACK

BOOL CALLBACK **A3DENUMCALLBACK**(
    LPGUID lpGuid,
    LPCSTR lpcstrDescription,
    LPCSTR lpcstrModule,
    LPVOID lpContext
);

Description

This callback is used in conjunction with the **IA3d5::A3dEnumerate** function. Its purpose is to enumerate all available sound devices on the system, passing a unique identifier and a textual description of each. After the call to A3dEnumerate, the A3D 3.0 engine calls A3DENUMCALLBACK once for each sound device. It is up to the application to remember all or some of the GUID's and then choose the desired one. Once a GUID has been determined, it is passed into **IA3d5::Init** or **IA3d5::InitEx**, and the A3D 3.0 engine will initialize and render using that device.

Arguments

LPGUID lpGuid	Unique identifier for this device.
LPCSTR lpcstrDescription	Pointer to a NULL terminated string that describes this device.
LPCSTR lpcstrModule	Pointer to a NULL terminated string that describes the driver name for this device.
LPVOID lpContext	A 32-bit pointer, determined by the application, that is passed into each call to A3DENUMCALLBACK unchanged. This is useful for keeping state across calls and for accessing global data.

Return Values

Return **True** to continue enumerating, **False** to stop.

IA3dListener Interface

A3D is rendered from the perspective of a listener, just as graphics are displayed from a viewpoint. The position and orientation of the listener determines how the scene ultimately sounds and the **IA3dListener** interface provides the methods for controlling the listener parameters.

A3D creates a listener at initialization. The application gains access to it by querying for the interface in the following way:

```
/* IA3d5 already exists... */
hr = pA3d5->QueryInterface(IID_IA3DLISTENER, &pListener);
```

The variable *pListener* is a pointer to the listener interface and all listener methods are accessed through this.

The listener has 3 properties:

- Position
- Orientation
- Velocity

There are two methods for setting each property; the only difference between them being the data type passed:

- Send values of a property individually (three numbers to represent the X, Y, and Z coordinates)
- Send values together in an array

To set the position of the listener, use:

```
pListener->SetPosition3f(fLisX, fLisY,fLisZ);
```

To set the orientation of the listener, use:

```
pListener->SetOrientation6f(fLisDirX, fLisDirY, fLisDirZ,
         fLisUpX, fLisUpY, fLisUpZ);
```

Remember that the two vectors should be unit vectors and perpendicular to each other.

To set the velocity of the listener, use:

```
pListener->SetVelocity3f(fLisVelX, fLisVelY, fLisVelZ);
```

As with all linear measurements in A3D, the default units here are metric (meters for distance and meters per second for velocity).

It's worth noting at this point that the ultimate results of calls to these three methods can be modified by a transformation matrix. This is discussed in Chapter 6: “Geometry Engine Reference Pages”. For now, when dealing with direct path only, matrices can be ignored and the methods set up the listener exactly in accordance with the values sent to them.

IA3dListener::AddRef

Increments the IA3dListener reference count.

Prototype

ULONG **AddRef**(
    void
);

Parameters

None.

Return Values

Returns the new reference count.

Description

When going through a COM method such as **QueryInterface** or **NewSource** to get an interface pointer to a component, the reference count of the component is automatically incremented. The reference count is used to let the component know when nothing is accessing it anymore and that it can delete itself from memory.

Whenever an interface pointer is assigned to another interface pointer, the **AddRef** method should be called to let the component know that two pointers are using the same interface. Now when the **Release** method is called, the component won't delete itself since it has been told something else is still using it. Consider the following example:

```
hr = pRoot->QueryInterface(IID_IBox, (void **)&pBox1);
if (SUCCEEDED(hr))
{
   pBox1->DrawIt();
   pBox2 = pBox1;
   pBox2->AddRef();
   pBox1->Release();
}
```

While *pBox1* is now invalid because it has been released, *pBox2* remains intact and can still be used.
All A3D 3.0 interfaces inherit the COM **IUnknown** interface, which contains the methods **AddRef**, **QueryInterface**, and **Release**. *Inside COM* by Microsoft Press is an excellent resource for detailed information on COM.

See Also

IA3dListener::QueryInterface
IA3dListener::Release

IA3dListener::QueryInterface

Returns an interface pointer for a supported interface.

Prototype

```
HRESULT QueryInterface(
    REFIID          iid,
    LPVOID FAR      *pInterface
);
```

Parameters

iid	Interface identifier. Specify only IID_IA3dListener.
pInterface	Address of a pointer to an interface which will be filled out by the method

Return Values

S_OK
E_NOINTERFACE

Description

All A3D interfaces inherit the **IUnknown** interface that contains a method called **QueryInterface**. This method is used to let the application know what other interfaces a particular interface supports, and to return a pointer to a requested interface if it is supported. The different A3D interfaces support different interfaces.

The **IA3dListener** interface doesn't support any other interfaces, so the only valid value for *iid* is IID_IA3dListener, which will return another listener interface pointer and increment the reference count.

Calling any **QueryInterface** and asking for an interface that isn't supported will return the error E_NOINTERFACE. The address of the pointer passed in to the method will be left at the value it was set to by the calling method, so it may not be NULL. For this reason, it is essential to check the return value of this method.

See Also

IA3dListener::AddRef
IA3dListener::Release

IA3dListener::Release

Decrements the IA3dListener reference count.

Prototype

```
ULONG Release(
    void
);
```

Parameters

None.

Return Values

Returns the new reference count.

Description

When going through a COM method such as **QueryInterface** or **NewSource** to get an interface pointer to a component, the reference count of the component is automatically incremented. The reference count is used to let the component know when nothing is accessing it anymore and that it can delete itself from memory.

Calling **IA3dListener::Release** decrements the reference count for the **IA3dListener** interface, and if it is 0, the object deletes itself from memory.

Note that **IA3d5**, **IA3dGeom2**, and **IA3dListener** all share the same reference count, as they are simply different interfaces onto the same base object. Only when all three have been released will the reference count of any one of them be 0.

All A3D 3.0 interfaces inherit the COM **IUnknown** interface that contains the methods **AddRef**, **QueryInterface**, and **Release**. *Inside COM* by Microsoft Press is an excellent resource for detailed information on COM.

See Also

IA3dListener::AddRef
IA3dListener::QueryInterface

IA3dListener::Set/GetOrientation

Sets and gets the orientation of the listener.

Prototype

HRESULT **SetOrientation6f**(
    A3DVAL *fFrontX*, A3DVAL *fFrontY*, A3DVAL *fFrontZ*,
    A3DVAL *fUpX*, A3DVAL *fUpY*, A3DVAL *fUpZ*
);

HRESULT **SetOrientation6fv**(
    A3DVAL **fFrontXYZUpXYZ*
);

HRESULT **GetOrientation6f**(
    A3DVAL **fFrontX*, A3DVAL **fFrontY*, A3DVAL **fFrontZ*,
    A3DVAL **fUpX*, A3DVAL **fUpY*, A3DVAL **fUpZ*
);

HRESULT **GetOrientation6fv**(
    A3DVAL **fFrontXYZUpXYZ*
);

Parameters

fFrontX, *fFrontY*, *fFrontZ* Three dimensional vector defining the front direction.

fUpX, *fUpY*, *fUpZ* Three dimensional vector defining the up direction.

fFrontXYZUpXYZ Array of six A3DVALs describing the front and up directions.

Return Values

S_OK

Description

IA3dListener::SetOrientation sets the orientation of the listener in 3D space. The parameters it takes are two perpendicular vectors defining the forward and up directions for the listener.

The orientation described by these vectors is relative to the transformation applied to the listener. If **IA3dGeom2::BindListener** is not being used, there is no transformation applied to the listener so the vectors are absolute. If **IA3dGeom2::BindListener** is being used, the vectors are relative to the coordinate system described by the current matrix when **IA3dGeom2::BindListener** was called.

Using this method to set the listener orientation will override the results of using **IA3dListener::SetOrientationAngles**. The two methods simply use different inputs to perform the same function.

IA3dListener::GetOrientation returns the orientation vectors of the listener relative to its coordinate system. If **IA3dListener::SetOrientationAngles** has been used, the vectors will have been computed from the the angles supplied to that method, so those vectors will be returned rather than the last set of vectors sent to **IA3dListener::SetOrientation**. Unlike angles, using vectors to describe an orientation is deterministic — for a given orientation there is only one set of front and up vectors that define that orientation.

See Also

IA3dListener::Set/GetOrientationAngles
IA3dGeom2::BindListener

IA3dListener::Set/GetOrientationAngles

Sets and gets the orientation of the listener.

Prototype

HRESULT **SetOrientationAngles3f**(
A3DVAL *fHeading*, A3DVAL *fPitch*, A3DVAL *fRoll*,
);

HRESULT **SetOrientationAngles3fv**(
A3DVAL **fHPR*,
);

HRESULT **GetOrientationAngles3f**(
A3DVAL **pHeading*, A3DVAL **fPitch*, A3DVAL **fRoll*,
);

HRESULT **GetOrientationAngles3fv**(
A3DVAL **fHPR*,
);

Parameters

fHeading, *fPitch*, *fRoll* Euler angles describing the orientation of the listener.

fHPR Array of three A3DVALs describing heading, pitch and roll.

Return Values

S_OK

Description

IA3dListener::SetOrientationAngles sets the orientation of the listener in 3D space. The parameters it takes are rotation values in degrees. *fHeading* represents rotation around the Y (up) axis, *fPitch* rotation about the X (right) axis, and *fRoll* rotation about the Z (out) axis. The rotations are applied in the following order: *fHeading*, *fPitch*, and *fRoll*.

The rotation described by the three angles is relative to the transformation applied to the listener. If **IA3dGeom2::BindListener** is not being used, there is no transformation applied to the listener so the angles are absolute. If **IA3dGeom2::BindListener** is being used, the rotation is relative to the coordinate system described by the current matrix when **IA3dGeom2::BindListener** was called.

Using this method to set the listener orientation will override the results of using **IA3dListener::SetOrientation**. The two methods simply use different inputs to perform the same function.

IA3dListener::GetOrientationAngles returns the rotation angles of the listener relative to its coordinate system. If **IA3dListener::SetOrientation** has been used, the angles will have been computed from the two vectors supplied to that method, so those angles will be returned rather than the last set of angles sent to **IA3dListener::SetOrientationAngles**. It's worth noting that the same orientation can be described by more than one set of rotation angles. For example, heading, pitch and roll of 0, 0, 0 is the same as 180, 180, 180. If the orientation was set using the vector method, **IA3dListener::GetOrientationAngles** might not return the most obvious angles for that orientation although they will be correct.

See Also

IA3dListener::Set/GetOrientation
IA3dGeom2::BindListener

IA3dListener::Set/GetPosition

Sets and gets the direction of the listener.

Prototype

HRESULT **SetPosition3f**(
    A3DVAL *fx*, A3DVAL *fy*, A3DVAL *fz*
);

HRESULT **SetPosition3fv**(
    A3DVAL **fxyz*
);

HRESULT **GetPosition3f**(
    LPA3DVAL *fx*, LPA3DVAL *fy*, LPA3DVAL *fz*
);

HRESULT **GetPosition3fv**(
    LPA3DVAL *fxyz*
);

Parameters

fx, *fy*, *fz*	Three A3DVALs for the position of the listener.
fxyz	An array of three A3DVALs for the position of the listener.

Return Values

S_OK

Description

IA3dListener::SetPosition sets the location of the listener in 3D space. The parameters it takes specify the location of the listener in three-dimensional space. If **IA3dGeom2::Listener** was used to apply a transformation to the listener, the coordinates specified in **IA3dListener::SetPosition** will be modified by that transformation. If the listener isn't bound to a matrix, the position set by this method will be in absolute world coordinates.

See Also

IA3dListener::Set/GetOrientation
IA3dGeom2::BindListener

IA3dListener::Set/GetVelocity

Sets and gets the velocity of the listener.

Prototype

HRESULT **SetVelocity3f**(
    A3DVAL *fvx*, A3DVAL *fvy*, A3DVAL *fvz*
);

HRESULT **SetVelocity3fv**(
    A3DVAL **fvxyz*
);

HRESULT **GetVelocity3f**(
    A3DVAL **fvx*, A3DVAL **fvy*, A3DVAL **fvz*
);

HRESULT **GetVelocity3fv**(
    A3DVAL **fvxyz*
);

Parameters

fvx, *fvy*, *fvz*	Three A3DVALs for the velocity vector of the source.
fvxyz	An array of three A3DVALs for the velocity vector of the source.

Return Values

S_OK

Description

IA3dListener::SetVelocity sets the velocity vector of the listener. The speed of the listener is determined from the length of this vector, and the direction of its motion from the direction of the vector. This information is used to compute Doppler shift.

The velocity vector is transformed by the listener matrix if **IA3dGeom2::BindListener** was used, otherwise it is in absolute world coordinates.

See Also

None.

IA3dReverb Interface

A3D 3.0 reverb presets and custom reverbs take the form of IA3dReverb objects. These objects are created using **IA3d5::NewReverb**, then modified using their member functions (such as **IA3dReverb::SetReverbPreset**). Finally, reverb is enabled by binding a reverb to the current environment via the **IA3d5::BindReverb** call.

IA3dReverb objects can be bind and unbound as desired. To stop all reverb, bind NULL. To replace the current reverb with a new one, either modify the current bound reverb object or bind a new IA3dReverb object. There is no limit to the number of reverb objects that can exist at any one time, but only one reverb can be bound at any time. In the future, it may be possibly to bind multiple reverbs.

IA3dReverb::AddRef

Increments the IA3dReverb reference count.

Prototype

ULONG **AddRef**(
    void
);

Parameters

None.

Return Values

Returns the new reference count.

Description

When going through a COM method such as QueryInterface or NewReverb to get an interface pointer to a component, the reference count of the component is automatically incremented. The reference count is used to let the component know when nothing is accessing it anymore and that it can delete itself from memory.

Whenever an interface pointer is assigned to another interface pointer, the AddRef method should be called to let the component know that two pointers are using the same interface. Now when the Release method is called, the component won't delete itself since it has been told something else is still using it. Consider the following example:

```
hr = pRoot->QueryInterface(IID_IBox, (void **)&pBox1);
if (SUCCEEDED(hr))
{
   pBox1->DrawIt();
   pBox2 = pBox1;
   pBox2->AddRef();
   pBox1->Release();
}
```

While pBox1 is now invalid because it has been released, pBox2 remains intact and can still be used.

All A3D 3.0 interfaces inherit the COM IUnknown interface that contains the methods AddRef, QueryInterface, and Release. "Inside COM" by Microsoft Press is an excellent resource for detailed information on COM.

See Also

IA3d5::QueryInterface

IA3d5::Release

IA3dReverb::QueryInterface

Returns an interface pointer for a supported interface.

Prototype

```
HRESULT QueryInterface(
    REFIID          iid,
    LPVOID FAR      *ppInterface
);
```

Parameters

iid	Interface identifier. Specify only IID_IA3dReverb.
ppInterface	Address of a pointer to an interface. The function fills out the pointer.

Return Values

S_OK
E_NOINTERFACE

Description

All A3D interfaces inherit the IUnknown interface that contains a method called QueryInterface. This method is used to let the application know what other interfaces a particular interface supports, and to return a pointer to a requested interface if it is supported. The different A3D interfaces support different interfaces.

The IA3dReverb interface doesn't support any other interfaces, so the only valid value for iid is IID_IA3dReverb, which will return another Reverb interface pointer and increment the reference count.

Calling any QueryInterface and asking for an interface that isn't supported will return the error E_NOINTERFACE. The address of the pointer passed in to the method will be left at the value it was set to by the calling method, so it may not be NULL. For this reason, it is essential to check the return value of this method.

See Also

IA3d5::AddRef

IA3d5::Release

IA3dReverb::Release

Decrements the IA3dReverb reference count.

Prototype

```
ULONG Release(
    void
);
```

Parameters

None.

Return Values

Returns the new reference count.

Description

When going through a COM method such as QueryInterface or NewReverb to get an interface pointer to a component, the reference count of the component is automatically incremented.

The reference count is used to let the component know when nothing is accessing it anymore and that it can delete itself from memory.

Calling IA3dReverb::Release decrements the reference count for the IA3dReverb interface, and if it is 0, the object deletes itself from memory. Typically, an application will not manually increment the reference count of an IA3dReverb interface, so **IA3dReverb::Release** will delete the Reverb.

All A3D 2.0 interfaces inherit the COM IUnknown interface that contains the methods AddRef, QueryInterface, and Release. “Inside COM” by Microsoft Press is an excellent resource for detailed information on COM.

Note: Though the preferred way of freeing reverb objects (and all A3d objects) is to call the COM Release method, the **IA3d5::Shutdown** function will actually release and free memory for all reverbs objects created.

See Also

IA3d5::AddRef
IA3d5::QueryInterface

IA3dReverb::Set/GetAllProperties

Sets or gets all properties of a custom reverb, or modifies all preset properties at once.

Prototype

HRESULT **Set/GetAllProperties**(
    A3DREVERB_PROPERTIES **pAllProperties*
);

Parameters

pAllProperties A pointer to an A3DREVERB_PROPERTIES structure which contains all the property values that describe this reverb.

Return Values

S_OK
A3DERROR_INVALID_ARGUMENT

Description

The SetAllProperties function takes a pointer to an A3DREVERB_PROPERTIES structure, which contains the property values that describe this reverb. The structure should be filled out with the appropriate property values before calling this function. If any of the values in the structure are out of range, A3DERROR_INVALID_ARGUMENT is returned.

GetAllProperties take a pointer to an A3DREVERB_PROPERTIES structure and fills it with the current values. Before calling this function, the dwSize field must be filled in with the proper size of the structure you are requesting values for (see below), but all other fields are ignored and overwritten.

See Also

A3DREVERB_PROPERTIES

A3DREVERB_CUSTOM

A3DREVERB_PRESET

IA3dReverb::Set/GetPresetDamping

Sets or gets the damping factor for this reverb object's active preset.

Prototype

```
HRESULT SetPresetDamping(
    A3DVAL fDamping,
);

HRESULT GetPresetDamping(
    A3DVAL *pfDamping,
);
```

Parameters

fDamping	Reverberation decay time
pfDamping	Pointer to be filled in with the reverberation decay time

Return Values

S_OK
A3DERROR_INVALID_ARGUMENT

Description

The SetPresetDamping function sets the damping factor for this reverb object's active preset.

The Damping parameter controls how fast high frequencies decay relative to middle and low frequencies. This value can range from 0.1f to 100.0f. Lower values shorten the high frequency decay time, resulting in a muffled effect. Higher values lengthen the high frequency decay value, resulting in a brighter effect. The default value depends on the preset chosen; use GetPresetDamping if you wish to make modifications based on a known value.

The new value will become active after the next call to **IA3d5::Flush**.

Note: This function only works when a reverb preset is currently active. No change will be heard when a custom reverb is in use.

See Also

None.

IA3dReverb::Set/GetPresetDecayTime

Sets or gets the decay time for this reverb object's active preset.

Prototype

HRESULT **SetPresetDecayTime**(
    A3DVAL *fDecayTime*,
);

HRESULT **GetPresetDecayTime**(
    A3DVAL **pfDecayTime*,
);

Parameters

fDecayTime Reverberation decay time

pfDecayTime Pointer to be filled in with the reverberation decay time

Return Values

S_OK
A3DERROR_INVALID_ARGUMENT

Description

The SetPresetDecayTime function sets the decay time for this reverb object.

The fDecayTime parameter controls the number of seconds it takes for the reverb to diminish by 60db, and is used to simulate the acoustical properties of a room. Longer times produce a very "live" room, with lots of reflections, while shorter times will make for a more "dead" room, down to where there is little noticeable reverb effect. This value is in seconds, and can range 0.1f to 100.0f. The default value depends on the preset chosen; use GetPresetDecayTime if you wish to make modifications based on a known value.

The decay time value is the number of seconds it takes for the reverb to diminish by 60db. Longer times simulate a very "live" room, while shorter times

The new value will become active after the next call to **IA3d5::Flush**.

Note: This function only works when a reverb preset is currently active. No change will be heard when a custom reverb is in use.

See Also

None.

IA3dReverb::Set/GetPresetVolume

Sets or gets the volume for this reverb object's active preset.

Prototype

```
HRESULT SetPresetVolume(
    A3DVAL Volume,
);

HRESULT GetPresetVolume(
    A3DVAL *pVolume,
);
```

Parameters

A3DVAL *Volume* Reverberation decay time

Return Values

S_OK
A3DERROR_INVALID_ARGUMENT

Description

The SetPresetVolume function sets the volume for this reverb object, relative to the direct path.

The Volume parameter controls the volume of the entire reverb effect, leaving intact the relative effect for individual sources. This value can range from 0.0f (no reverb effect) to 1.0f (maximum reverb effect). The default value depends on the preset chosen; use GetPresetVolume if you wish to make modifications based on a known value.

The new value will become active after the next call to **IA3d5::Flush**.

Note: This function only works when a reverb preset is currently active. No change will be heard when a custom reverb is in use.

See Also

None.

IA3dReverb::Set/GetReverbPreset

Selects a reverb preset for this reverb object, or returns the current preset.

Prototype

HRESULT **SetReverbPreset**(
    DWORD *preset*
);

HRESULT **GetReverbPreset**(
    DWORD **pPreset*
);

Parameters

Preset - (These are directly mapped to EAX environment types):

A3DREVERB_PRESET_GENERIC - The default when the reverb is created
A3DREVERB_PRESET_PADDEDCELL
A3DREVERB_PRESET_ROOM
A3DREVERB_PRESET_BATHROOM
A3DREVERB_PRESET_LIVINGROOM
A3DREVERB_PRESET_STONEROOM
A3DREVERB_PRESET_AUDITORIUM
A3DREVERB_PRESET_CONCERTHALL
A3DREVERB_PRESET_CAVE
A3DREVERB_PRESET_ARENA
A3DREVERB_PRESET_HANGAR
A3DREVERB_PRESET_CARPETEDHALLWAY
A3DREVERB_PRESET_HALLWAY
A3DREVERB_PRESET_STONECORRIDOR
A3DREVERB_PRESET_ALLEY
A3DREVERB_PRESET_FOREST
A3DREVERB_PRESET_CITY
A3DREVERB_PRESET_MOUNTAINS
A3DREVERB_PRESET_QUARRY
A3DREVERB_PRESET_PLAIN
A3DREVERB_PRESET_PARKINGLOT
A3DREVERB_PRESET_SEWERPIPE
A3DREVERB_PRESET_UNDERWATER
A3DREVERB_PRESET_DRUGGED

A3DREVERB_PRESET_DIZZY
A3DREVERB_PRESET_PSYCHOTIC

Return Values

S_OK

Description

The SetReverbPreset function selects a reverb preset. The selected preset can be modified with the advanced property functions (defined below). If the reverb preset is changed after having been modified, all modifications are lost, and the new preset is immediately enabled. An A3DReverb object that is currently bound to the world using **IA3d5::BindReverb** will continue to be bound even when the reverb preset is changed or the properties are changed. Any changes made to a bound A3DRreverb object will be applied to the world when the next **IA3d5::Flush** call is made. If a bound A3DReverb object is released, it will effectively be unbound right before it is deleted.

GetReverbPreset takes a pointer to a DWORD that will be filled with the current preset.

Note that switching between presets, or changing any reverb properties, will take place at the next call to **IA3d5::Flush**, and will happen smoothly. It's also important to remember that while WAV and MP3 files can have reverb effects applied to them, AC-3 tracks cannot.

See Also

None.

A3DREVERB_CUSTOM

Members

DWORD *dwSize*	Size of the structure. This must be set before passes a blank structure into any other API calls.
LONG *lRoom*	Controls the level of the room effect. This value can be between -10000 (minimum room effect) and 0 (maximum room effect).
LONG *lRoomHF*	Controls the attenuation at high frequencies relative to the intensity at low frequencies. This value can be between -10000 (no attenuation) and 0 (max attenuation).
FLOAT *flRoomRolloffFactor*	Controls the rolloff of room effect intensity vs. distance. This value can be between 0.0f and 10.0f. At 0.0f, reverberation intensity does not depend on source-listener distance. As this value is increased, the reverb decays faster with respect to the distance from the source.
FLOAT *flDecayTime*	Controls the reverberation decay time at low frequencies. This value is in seconds, and can range from 0.1f to 20.0f.
FLOAT *flDecayHFRatio*	Controls the ratio of high-frequency decay time relative to low-frequency decay time. This value can be between 0.1f (high frequencies decay faster than low frequencies) and 2.0f (high and low frequencies decay at the same rate).
LONG *lReflections*	Controls the intensity level of early reflections (relative to Room value). This value can be between -10000 and 1000. At –10000 no early reflections will be heard, increasing this value results in an increase in the volume of early reflections relative to late reflections.
FLOAT *flReflectionsDelay*	Controls the delay time of the first reflection (relative to the direct path). This value is in seconds, and can range from 0.0f to 0.3f.

LONG *lReverb*	Controls the intensity of late reverberation (relative to Room value). This value can be between -10000 (no reverberation) and 2000 (maximum reverberation).
FLOAT *flReverbDelay*	Defines the time limit between the early reflections and the late reverberation (relative to the time of the first reflection). This value is in seconds, and can range from 0.0f to 0.1f.
FLOAT *flDiffusion*	Controls the modal diffusion in the late reverberation decay. This value is a percentage, and can range from 0.0f (minimum diffusion) to 1.0 (maximum diffusion).
FLOAT *flDensity*	Controls the echo density in the late reverberation decay. This value is a percentage, and can range from 0.0f (minimum density) to 1.0 (maximum density).
FLOAT *flHFReference*	Controls the reference high frequency. This value is in kHz, and can range from 20.0f to 20000.0f. All low-pass effects are specified as high-frequency attenuation in dB relative to low frequencies, taking the same reference high frequency for all effects.

A3DREVERB_PRESET

Description

The A3DREVERB_PRESET structure specifies a predefined reverb with a couple of tweak values, described below.

Members

DWORD *dwSize*	Size of the structure. This must be set before passes a blank structure into any other API calls.
DWORD *dwEnvPreset*	Specify an A3D_REVERB_PRESET value.
A3DVAL *fVolume*	Overall volume of the reverb - any value between 0.0 (silence) and 1.0 (full effect).
A3DVAL fDecayTime	Seconds in decay (0 – 100.f)
A3DVAL *fDamping*	Damp value from (0 – 1)

A3DREVERB_PROPERTIES

The A3DREVERB_PROPERTIES structure is, in fact, a union. As preset reverbs contain different properties than custom reverbs, it is necessary to fill out the proper member structure before making these calls, else an error is returned. The dwType field must be properly specified, along with the dwSize fields of both the member structure and the A3DREVERB_PROPERTIES structure itself.

Modifying Presets

The Set/GetReverbProperties functions are available for presets merely for the convenience of being able to set all of the preset values at once. These values are identical to the SetPreset functions described elsewhere.

To use SetReverbProperties with a preset, specify a dwType of A3DREVERB_TYPE_PRESET, fill in the A3DREVERB_PRESET member structure with the same values as you would use when calling the individual functions, and call this function. The new values will become active after the next call to **IA3d5::Flush**.

To use GetReverbProperties with a preset, specify a dwType of A3DREVERB_TYPE_PRESET and call this function. The A3DREVERB_PRESET member structure will be filled in with the values that are currently in use. An error will be returned if the current reverb is not using a preset.

Creating Custom reverbs

The SetReverbProperties function allows the creation of custom reverbs, with all properties available for tweaking.

To use SetReverbProperties to create a custom preset, specify a dwType of A3DREVERB_TYPE_CUSTOM, completely fill in the A3DREVERB_CUSTOM member structure and call this function. The new values will become active after the next call to **IA3d5::Flush**.

To use GetReverbProperties with a custom reverb, specify a dwType of A3DREVERB_TYPE_CUSTOM and call this function. The A3DREVERB_CUSTOM member structure will be filled in with the values that are currently in use. An error will be returned if the current reverb is not using a custom reverb.

Members

DWORD *dwSize* — Size of the A3DREVERB_PROPERTIES structure. This must be set before this structure is passed into a function.

DWORD *dwType*	Either A3DREVERB_TYPE_PRESET or A3DREVERB_TYPE_CUSTOM, depending on which member structure of the union is being used.

The union:

A3DREVERB_PRESET *Preset*	Struct with information about the current reverb preset.

Or:

A3DREVERB_CUSTOM *Custom*	Struct describing the current user-defined reverb.

See Also

A3DREVERB_CUSTOM
A3DREVERB_PRESET

IA3dSource2 Interface

Sources are objects that make sound. A3D lets an application create as many sources as it needs and position them in 3D space. Each source is rendered from the perspective of the listener.

The **IA3dSource2** interface contains the methods used to manipulate a source. The application obtains **IA3dSource2** automatically when a new source is created. Sources are created by calling the **NewSource** method in **IA3d5**. A pointer to the interface is returned in a parameter to the method, as shown in this example:

```
/* IA3d5 already exists.... */

pA3d5->NewSource(A3DSOURCE_TYPE3D, &pSource_0);
```

In this case, *pSource_0* is the pointer to the **IA3dSource2** interface. (Since this is only an example, the return codes are ignored.)

There are two types of sources: *3D* and *native*. 3D sources can be positioned in the world and interact with any geometry being rendered, while native sources can be panned left and right and are not affected by distance or geometry.

Before a source can make any sound, it needs wave data to play back. There are several ways to get the wave data attached to the source but the simplest is to use **IA3dSource2::LoadWaveFile**:

```
pSource->LoadWaveFile("my_waves/heli.wav");
```

With wave data loaded, the source is ready to be played. The following code plays the source, moves it from one position to another, and then stops it:

```
pSource->Play(A3D_LOOPED);

for (fX = -10.0f; fX < 10; fX += 0.01f)
      pSource->SetPosition(fX, 0.0f, -5.0f);

pSource->Stop();
```

As with the listener, the source properties can be manipulated by a transformation matrix. This topic is discussed in “Transformation Matrix” on page 17. More information can also be found in Chapter 6: “Geometry Engine Reference Pages”.

IA3dSource2::AddRef

Increments the IA3dSource2 reference count.

Prototype

ULONG **AddRef**(
    void
);

Parameters

None.

Return Values

Returns the new reference count.

Description

When going through a COM method such as **QueryInterface** or **NewSource** to get an interface pointer to a component, the reference count of the component is automatically incremented. The reference count is used to let the component know when nothing is accessing it anymore and that it can delete itself from memory.

Whenever an interface pointer is assigned to another interface pointer, the **AddRef** method should be called to let the component know that two pointers are using the same interface. Now when the **Release** method is called, the component won't delete itself since it has been told something else is still using it. Consider the following example:

```
hr = pRoot->QueryInterface(IID_IBox, (void **)&pBox1);
if (SUCCEEDED(hr))
{
   pBox1->DrawIt();
   pBox2 = pBox1;
   pBox2->AddRef();
   pBox1->Release();
}
```

While *pBox1* is now invalid because it has been released, *pBox2* remains intact and can still be used.

All A3D 3.0 interfaces inherit the COM **IUnknown** interface that contains the methods **AddRef**, **QueryInterface**, and **Release**. *Inside COM* by Microsoft Press is an excellent resource for detailed information on COM.

See Also

IA3dSource2::Release
IA3dSource2::QueryInterface

IA3dSource2::AllocateAudioData

Allocates data for a sound source. You must set the format of the sound source before calling this function.

Prototype

HRESULT **AllocateAudioData**(
    INT *nSize*
);

Parameters

nSize Size of the memory data in bytes.

Return Values

S_OK

E_INVALIDARG if *nSize* < 1
A3DERROR_NEEDS_FORMAT_INFORMATION
A3DERROR_FAILED_CREATE_SOUNDBUFFER

Description

IA3dSource2::AllocateAudioData allocates memory for the source wave data and attempts to allocate the resources necessary to handle playing the source. Following this call, IA3dSource2::Lock should be used to lock the entire wave data space for writing and the wave data copied into the source using memcopy.

This method may fail for an unmanaged source if insufficient resources are available to support the requested format. For this reason it is essential to check the return value using the standard COM SUCCEEDED macro.

IA3dSource2::FreeAudioData is used to free the memory and resources allocated by **IA3dSource2::AllocateAudioData**. **IA3dSource2::Release** automatically frees all the resources assigned to the source.

See Also

IA3dSource2::Set/GetAudioFormat
IA3dSource2::Lock
IA3dSource2::Unlock

IA3dSource2::ClearPlayEvents

Clears all wave events for a source.

Prototype

HRESULT **ClearPlayEvents**(
    void
);

Parameters

None.

Return Values

S_OK

Description

This method clears all the playback position wave events set using **IA3dSource2::SetPlayEvent**.

See Also

IA3dSource2::SetPlayEvent

IA3dSource2::FreeAudioData

Releases allocated data for a sound source.

Prototype

HRESULT **FreeAudioData**(
    void
);

Parameters

None.

Return Values

S_OK
A3DERROR_NO_WAVE_DATA

Description

IA3dSource2::LoadFile and **IA3dSource2::AllocateAudioData** both allocate resources to play back the wave data of the source. **IA3dSource2::FreeAudioData** releases those resources and leaves the source effectively empty.

The resources that are freed are the memory used to store the wave data and the playback channel in the audio renderer.

See Also

IA3dSource2::AllocateAudioData
IA3dSource2::LoadFile

IA3dSource2::GetAudibility

Gets the calculated audibility of the source.

Prototype

HRESULT **GetAudibility**(
    LPA3DVAL *fAudibility*
);

Parameters

fAudibility Pointer to an A3DVAL which will be filled out by the method.

Return Values

S_OK

Description

A3D internally computes an audibility value for each source. This audibility is a function of gain and any frequency dependent attenuation caused by distance, occlusions, and any explicit settings from the application - the higher the value, the more audible the source.

IA3dSource2::GetAudibility is used to return that value to the application where it can be used to make decisions about, for example, whether to continue playing the source. The value is one frame old, as audibility can only be computed after all processing on the source has been completed inside the call to **IA3d5::Flush**.

Valid returned values for *fAudibility* are from 0.0 to 1.0 inclusive, 1.0 being fully occluded. If the source is not playing its audibility will be 0.0.

See Also

IA3d5::Flush
IA3dSource2::Set/GetGain
IA3dSource2::Set/GetPanValues
IA3dSource2::GetOcclusionFactor

IA3dSource2::GetOcclusionFactor

Gets the occlusion factor of the source.

Prototype

HRESULT **GetOcclusionFactor**(
    LPA3DVAL *fOcclusionFactor*
);

Parameters

fOcclusionFactor Pointer to an A3DVAL which will be filled out by the method.

Return Values

S_OK
A3DERROR_SOURCE_IN_NATIVE_MODE

Description

When processing geometry, A3D computes a value that specifies how much a source is occluded by surfaces. This isn't a gain value, as it doesn't take surface material properties into account, but simply a value that says how much of the source is blocked by a surface. **IA3dSource2::GetOcclusionFactor** is used to pass that value back to the application where it can be used to make decisions about what to do with the source. For example, if a source is fully occluded, its priority could be reduced because most likely the object making the sound isn't visible.

Valid returned values for *fOcclusionFactor* are from 0.0 to 1.0 inclusive, 1.0 being fully occluded. If the source is not playing its occlusion factor will be 0.0.

See Also

None.

IA3dSource2::GetStatus

Gets the activity status of the source.

Prototype

HRESULT **GetStatus**(
    LPDWORD *dwStatus*
);

Parameters

dwStatus Pointer to a DWORD, filled out by the method.

Return Values

S_OK

Description

This method returns the playback status of a source. The status specifies whether the source is playing, stopped, or has been requested for playing but is still waiting for **IA3d5::Flush** to be called. *dwStatus* is a bitmask of the following values: A3DSTATUS_PLAYING, A3DSTATUS_LOOPING, A3DSTATUS_WAITING_FOR_FLUSH. If it is 0 then the source is stopped.

See Also

IA3d5::Flush
IA3dSource2::Play
IA3dSource2::Stop

IA3dSource2::GetType

Gets the type of the source.

Prototype

```
HRESULT GetType(
    LPDWORD dwType
);
```

Parameters

dwType Pointer to a DWORD which will be filled out by the method.

Return Values

S_OK

Description

When a source is created with **IA3d5::NewSource**, a type is defined for the source. **IA3dSource2::GetType** returns the value that was passed in when the source was created, or the type that was derived from the parent source when a duplicate was created.

dwType is filled out by the method and is a bitmask representing all the flags that were set in **IA3d5::NewSource**.

Note that the type of a source can never be changed. It can only be set when the source is created. It is possible to change the playback mode of some sources. For example, a resource managed A3D source can be played back as stereo even though it is of type A3D. This doesn't change its type — it will still be reported as an A3D source.

See Also

IA3d5::NewSource
IA3dSource2::Set/GetRenderMode

IA3dSource2::GetAudioSize

Gets the size of the audio data.

Prototype

HRESULT **GetAudioSize**(
    void
);

Parameters

None.

Return Values

The size of the Allocated Audio Buffer in bytes. 0 if no data is allocated.

Description

This method is used to find out how much memory is allocated to store the audio data for a source. It returns the number of bytes allocated, and may be 0 if there is no audio data associated with the source.

See Also

IA3dSource2::AllocateAudioData
IA3dSource2::FreeAudioData

IA3dSource2::LoadWaveData

Loads audio data from memory into the source.

Prototype

```
HRESULT LoadWaveData(
    LPVOID    pvWaveData
    DWORD     dwSize
);
```

Parameters

pvWaveData Pointer to the audio data in memory.

dwSize Size of the data.

Return Values

S_OK
E_POINTER
E_INVALIDARG

Description

This method is used to load audio data from memory into a source. The data pointed to by *pvWaveData* must also contain the wave file header describing the format of the data to be loaded. It is equivalent to calling **IA3dSource2::SetFormat** and **IA3dSource2::AllocateAudioData**.

This method may fail for an unmanaged source if insufficient resources are available to support the requested format. For this reason it is essential to check the return value using the standard COM SUCCEEDED macro.

The memory allocated by this function is freed by **IA3dSource2::FreeAudioData**.

See Also

IA3dSource2::LoadFile
IA3dSource2::Set/GetAudioFormat
IA3dSource2::AllocateAudioData
IA3dSource2::FreeAudioData

IA3dSource2::LoadFile

Loads audio data from a file.

Prototype

HRESULT **LoadFile**(
    char **szFileName*,
    DWORD *dwFormat*
);

Parameters

szFileName Path and file name of wave data file to load.

dwFormat Specifies how to load this file.

A3DSOURCE_FORMAT_AUTO

A3DSOURCE_FORMAT_WAVE

A3DSOURCE_FORMAT_MP3

A3DSOURCE_FORMAT_AC3

A3DSOURCE_STREAMING

Return Values

S_OK
A3DERROR_FAILED_FILE_OPEN
A3DERROR_UNRECOGNIZED_FORMAT
A3DERROR_FAILED_ALLOCATE_WAVEDATA
A3DERROR_FAILED_LOCK_BUFFER
A3DERROR_FAILED_UNLOCK_BUFFER

Description

IA3dSource2::LoadFile is a convenience function which does all the I/O necessary to open an audio file and read it into memory. It also sets the source up to read the format of the wave data and store the samples in the source.

See Also

IA3dSource2::LoadWaveData
IA3dSource2::AllocateAudioData

IA3dSource2::Lock

Allows data to be written to a buffer.

Prototype

```
HRESULT Lock(
    DWORD       dwWriteCursor,
    DWORD       dwNumBytes,
    LPVOID      *pvAudioPtr1,
    LPDWORD     dwAudioBytes1,
    LPVOID      *pvAudioPtr2,
    LPDWORD     dwAudioBytes2,
    DWORD       dwFlags
);
```

Parameters

dwWriteCursor Offset from the start of the buffer.

dwNumBytes Number of bytes to lock.

pvAudioPtr1 Pointer to first block of available data.

dwAudioBytes1 Number of bytes in first block.

pvAudioptr2 Pointer to second block of available data.

dwAudioBytes2 Number of bytes in second block.

dwFlags Specifies the lock mode. Select one of:

A3D_FROMWRITECURSOR	Uses current cursor position and ignores *dwWriteCursor*.
A3D_ENTIREBUFFER	Ignores *dwWriteCursor* and *dwNumBytes* — locks the whole buffer.

Return Values

S_OK
A3DERROR_NO_WAVE_DATA
A3DERROR_FAILED_LOCK_BUFFER

Description

Use **IA3dSource2::Lock** to find a portion of the wave data in a sound source that can be safely written to — a block of data that is not currently being played. The wave data in a sound source is stored in a circular buffer so, depending on how close to the end of the buffer the write cursor is and how much data has been requested, a second pointer may be returned. Use **IA3dSource2::Unlock** after you are finished manipulating the data.

See Also

IA3dSource2::Unlock
IA3dSource2::LoadFile
IA3dSource2::AllocateAudioData

IA3dSource2::Play

Starts a sound source playing.

Prototype

```
HRESULT Play(
    INT         nMode
);
```

Parameters

mode	A3D_LOOPED or A3D_SINGLE. Determines whether the sound is played in a continuous loop or only once.

Return Values

S_OK
A3DERROR_NO_WAVE_DATA
A3DERROR_UNKNOWN_PLAYMODE
A3DERROR_FAILED_PLAY

Description

IA3dSource2::Play starts a sound source playing either once only or in looping mode. This doesn't start it playing immediately as all modifications to a source (other than changing the wave data) are deferred until **IA3d5::Flush** is called. This means if **IA3d5::Flush** is never called, no sources will ever be played.

Calling this method is not guaranteed to result in the source actually being played; it just lets the resource manager know that the application has requested to play this source. If resources are available, it will be played. If no resources are available, the resource manager will weigh the importance and audibility of this source against all the others the application has asked to play and decide at that point whether it should replace a currently playing source with this one. Resource management and re-allocation is done approximately every 10 ms, so in the situation where playing a source successfully displaces another source, there may be a small lag before the new source starts playing. Generally this latency will be much less than the length of a video frame and shouldn't be noticeable.

See Also

IA3dSource2::Stop
IA3d5::Flush
IA3dSource2::Set/GetPriority
IA3d5::Set/GetRMPriorityBias

IA3dSource2::QueryInterface

Returns an interface pointer for a supported interface.

Prototype

```
HRESULT QueryInterface(
    REFIID          iid,
    LPVOID FAR      *pInterface
);
```

Parameters

iid	Interface identifier. Specify only IID_IA3dSource2.
pInterface	Address of a pointer to an interface which will be filled out by the method.

Return Values

S_OK
E_NOINTERFACE

Description

All A3D interfaces inherit the **IUnknown** interface that contains a method called **QueryInterface**. This method is used to let the application know what other interfaces a particular interface supports, and to return a pointer to a requested interface if it is supported. The different A3D interfaces support different interfaces.

The **IA3dSource2** interface doesn't support any other interfaces, so the only valid value for *iid* is IID_IA3dSource2, which will return another source interface pointer and increment the reference count.

Calling any **QueryInterface** and asking for an interface that isn't supported will return the error E_NOINTERFACE. The address of the pointer passed in to the method will be left at the value it was set to by the calling method, so it may not be NULL. For this reason, it is essential to check the return value of this method.

See Also

IA3dSource2::AddRef
IA3dSource2::Release

IA3dSource2::Release

Decrements the IA3dSource2 reference count.

Prototype

```
ULONG Release(
    void
);
```

Parameters

None.

Return Values

Returns the new reference count.

Description

When going through a COM method such as **QueryInterface** or **NewSource** to get an interface pointer to a component, the reference count of the component is automatically incremented. The reference count is used to let the component know when nothing is accessing it anymore and that it can delete itself from memory.

Calling **IA3dSource2::Release** decrements the reference count for the **IA3dSource2** interface, and if it is 0, the object deletes itself from memory. Typically, an application will not manually increment the reference count of an **IA3dSource2** interface, so **IA3dSource2::Release** will delete the source.

All A3D 3.0 interfaces inherit the COM **IUnknown** interface that contains the methods **AddRef**, **QueryInterface**, and **Release**. *Inside COM* by Microsoft Press is an excellent resource for detailed information on COM.

See Also

IA3d5::NewSource
IA3d5::DuplicateSource
IA3dSource2::AddRef
IA3dSource2::QueryInterface.

IA3dSource2::Rewind

Rewinds a sound source back to the beginning of the wave data.

Prototype

```
HRESULT Rewind(
    void
);
```

Parameters

None.

Return Values

S_OK

Description

Rewinds the playback cursor in a sound source to the beginning of the wave data. It is equivalent to calling **IA3dSource2::SetPlayPosition(0)**. This does not trigger the source to start playing — if it is playing it continues to play and if it is stopped it remains stopped.

See Also

IA3dSource2::Set/GetPlayPosition

IA3dSource2::Set/GetAudioFormat

Sets and gets the format of the wave information.

Prototype

HRESULT **SetAudioFormat**(
    LPVOID   *pWaveFormat*
);

HRESULT **GetAudioFormat**(
    LPVOID   *pWaveFormat*
);

Parameters

pWaveFormat Pointer to a WAVEFORMATEX structure, cast to a void pointer.

Return Values

S_OK
E_POINTER
E_OUTOFMEMORY
A3DERROR_CANNOT_CHANGE_FORMAT_FOR_ALLOCATED_BUFFER
A3DERROR_NO_WAVE_DATA (**GetAudioFormat** only)

Description

IA3dSource2::SetAudioFormat is used to specify the type of audio data which will be loaded into the source with a subsequent call to **IA3dSource2::AllocateAudioData**. It must be called before **IA3dSource2::AllocateAudioData** can be used.

On Win32 platforms, the parameter *pWaveFormat* is a pointer to a WAVEFORMATEX structure, cast to a void pointer. When using **IA3dSource2::SetAudioFormat**, this structure should be allocated and filled out by the application. The source object will create its own structure internally and copy the data from *pWaveFornat* into it so the application can free its copy. The data contained in the structure specifies properties such as the sample rate, resolution and number of channels in the wave data to be loaded. See the Win32 SDK reference for details on the WAVEFORMATEX structure.

Currently A3D only supports PCM data where

`PWaveFormat->wFormatTag = WAVE_FORMAT_PCM`

The field cbSize in the WAVEFORMATEX structure is ignored. Similarly **GetAudioFormat** only copies memory equal to `sizeof(WAVEFORMATEX)` and `not sizeof(WAVEFORMATEX) + pWaveFormat->cbSize`.

The format of a source can't be changed after any wave data has been copied into it. To change the format, the wave data must be freed using **IA3dSource2::FreeAudioData**.

See Also

IA3dSource2::AllocateAudioData

IA3dSource2::Set/GetCone

Sets the directionality of a source cone.

Prototype

```
HRESULT SetCone(
    A3DVAL   fInnerAngle,
    A3DVAL   fOuterAngle,
    A3DVAL   fGain
);

HRESULT GetCone(
    A3DVAL   *fInnerAngle,
    A3DVAL   *fOuterAngle,
    A3DVAL   *fGain
);
```

Parameters

fAngle1	Inner cone angle in degrees.
fAngle2	Outer cone angle in degrees.
fGain	Gain at *fOuterAngle*.

Return Values

S_OK

Description

IA3dSource2::SetCone is used to specify the cone angles for directional sound sources. The two angles, *fInnerAngle* and *fOuterAngle* define the size of the cone. Between 0 degrees and *fInnerAngle*, the source will be at the level specified in any call to **IA3dSource2::SetGain** (plus any effect caused by distance or occlusions). Between *fInnerAngle* and *fOuterAngle* the source gain is multiplied by the cone gain calculated by interpolating between 1.0 and *fGain* according to the bearing of the listener from the source. From *fOuterAngle* to 180 degrees, the source gain is multiplied by *fGain*.

Enabling a sound cone for a source results in a small performance overhead as some extra calculations need to be performed on the source. Setting either *fOuterAngle* to 0 or *fGain* to 1 disables cone processing and the source is treated as omnidirectional.

Cones only affect direct path gain and are ignored for reflections.

See Also

IA3dSource2::Set/GetDistanceModelScale

Changes the distance attenuation curve for a source.

Prototype

HRESULT **SetDistanceModelScale**(
    A3DVAL *fScale*
);

HRESULT **GetDistanceModelScale**(
    LPA3DVAL p*fScale*
);

Parameters

fScale Scale factor for the distance model. Valid values are 0.0 to infinity.

Return Values

S_OK
E_INVALIDARG

Description

Source audibility is reduced with distance from the listener. Two factors affect the rate of that attenuation — the minimum distance set for the source, and the scaling applied to the curve beyond the minimum distance. **IA3dSource2::SetDistanceModelScale** affects the latter.

By default, sources are attenuated by 6 dB for each doubling in distance. If the minimum distance is 1 m then at 2 m from the listener the source will be at -6 dB, at 4 m it will be at -12 dB, and at 8 m it will be -18 dB etc. Using this method to modify the curve does not affect the minimum distance. Instead, for the purpose of gain attenuation, it recalculates the range from the listener to the source in the following way:

$$new\_range = ((range - min\_dist) \times scale) + min\_dist$$

This has the effect of flattening or exaggerating the curve but without causing any discontinuity at the minimum distance. If the unmodified range of a source is within the minimum distance, distance attenuation is set to 0 and the new range calculation ignored.

See Also

IA3dSource2::Set/GetMinMaxDistance

IA3dSource2::Set/GetDopplerScale

Sets and gets the exaggerated Doppler effect on a source.

Prototype

HRESULT **SetDopplerScale**(
    A3DVAL *fDopplerScale*
);

HRESULT **GetDopplerScale**(
    LPA3DVAL *pfDopplerScale*
);

Parameters

fDopplerScale Multiplier for the Doppler effect.

Return Values

None.

Description

Doppler shift is the effect that the motion of a source and listener has on the perceived frequency of the sound made by the source. Sounds moving towards a listener are raised in pitch, and those moving away lowered in pitch. The amount of pitch change is proportional to the speed of the source and listener along the line that joins them. Speed along that line is computed from the velocity vectors of the source and listener.

Doppler shift on a source can be exaggerated or reduced using **IA3dSource2::SetDopplerScale**. The default is 1.0 (correct Doppler), while 2.0 doubles the effect and 0.5 reduces it by half. Depending on the original sample rate of the source there are limits to how much the Doppler effect can be exaggerated, but 0.5 – 2.0 are reasonable values. If input Doppler scale is less than zero, the Doppler scale is set to zero.

See Also

IA3d5::Set/GetDopplerScale
IA3dListener::Set/GetVelocity
IA3dSource2::Set/GetVelocity

IA3dSource2::Set/GetEq

Sets the tonal equalization of a source.

Prototype

```
HRESULT SetEq(
    A3DVAL          fHighFreq
);

HRESULT GetEq(
    LPA3DVAL        fHighFreq
);
```

Parameters

fHighFreq Floating point number between 0.0 and 1.0 inclusive (default 1.0).

Return Values

S_OK

E_INVALIDARG

Description

IA3dSource2::SetEq applies an equalization effect to the source. It is similar in effect to a treble control on a stereo system and is completely independent of distance and gain. It is low-pass only and doesn't allow high frequencies to be boosted.

If 0.0 < *fEq* < 1.0, high frequencies are attenuated more as *fEq* approaches 0.0. The default setting of 1.0 means there is no additional high frequency attenuation applied to sources.

This method is useful for simulating different environments. For example, *fEq* = 0.3 would make the source sound very muffled, as if it is underwater.

This EQ value is applied in addition to any EQ value specified globally for all sources. In common with all global/local scalars, *fEq* is multiplied by the global EQ value, which is set using **IA3d5::SetEq**.

See Also

IA3d5::SetEq

IA3dSource2::Set/GetGain

Sets and gets the playback gain of a source.

Prototype

```
HRESULT SetGain(
    A3DVAL  fGain,
);

HRESULT GetGain(
    LPA3DVAL        pfGain,
);
```

Parameters

fGain The gain of the source.

Return Values

S_OK

Description

fGain is in the range 0.0 to 1.0, where 0.0 is silence and 1.0 (the default) is 0 dB which is the maximum loudness for a source. Each reduction by half represents a 6 dB attenuation, so *fGain* = 0.5 is equivalent to -6dB, *fGain* = 0.25 equivalent to -12dB, *fGain* = 0.125 to -18dB and so on.

Setting the gain of a source sets the maximum possible volume that source will be played back at. Any attenuation due to distance or occlusions will be in addition to the attenuation explicitly set by this method.

This method is the local equivalent of **IA3d5::SetOutputGain** and the gain set here is multiplied with the global output gain to get the final gain of the source.

See Also

IA3d5::SetOutputGain

IA3dSource2::Set/GetMinMaxDistance

Sets and gets the range over which the distance model will be applied to a source.

Prototype

```
HRESULT SetMinMaxDistance(
    A3DVAL      fMinDistance,
    A3DVAL      fMaxDistance,
    DWORD       dwBehavior
);

HRESULT GetMinMaxDistance(
    LPA3DVAL    pfMinDistance,
    LPA3DVAL    pfMaxDistance,
    LPDWORD     pdwBehavior
);
```

Parameters

fMinDistance	Minimum distance value.
fMaxDistance	Maximum distance value.
dwBehavior	Behavior at max distance.

Return Values

S_OK

Description

This method allows the distance model for the source to be modified. *fMinDistance* is the distance from the listener that the source must go beyond before the distance the model starts to attenuate it. *fMaxDistance* is the maximum distance from the listener that the distance model will affect the source — beyond that the source will not be attenuated any more.

The value of *fMinDistance* shapes the attenuation curve. Sources are attenuated by 6 dB with each doubling in distance from the listener, but since this attenuation doesn't begin until the source has reached *fMinDistance*, the first reduction of 6 dB occurs at double the minimum distance. Moving the minimum distance further out reduces the attenuation rate, making sources audible at greater ranges, and bringing it in increases the attenuation rate.

The behavior at max distance is determined by the *dwBehavior* parameter. A3D_AUDIBLE is the default and causes the source to play at a constant gain once it is

further than the maximum distance. A3D_MUTE causes the source to mute when it reaches the max distance.

See Also

IA3dSource2::Set/GetDistanceModelScale
IA3d5::Set/GetDistanceModelScale

IA3dSource2::Set/GetOrientation

Sets and gets the direction of the sound source.

Prototype

HRESULT **SetOrientation6f**(
A3DVAL *fDirX*, A3DVAL *fDirX*, A3DVAL *fDirX*,
A3DVAL *fUpX*, A3DVAL *fUpX*, A3DVAL *fUpX*,
);

HRESULT **SetOrientation6fv**(
LPA3DVAL *fDirXYZUpXYZ*
);

HRESULT **GetOrientation6f**(
A3DVAL **fDirX*, A3DVAL **fDirX*, A3DVAL **fDirX*,
A3DVAL **fUpX*, A3DVAL **fUpX*, A3DVAL **fUpX*,
);

HRESULT **GetOrientation6fv**(
LPA3DVAL*fDirXYZUpXYZ*
);

Parameters

fDirX, *fDirY*, *fDirZ*, *fUpX*, *fUpY*, *fUpZ* Two perpendicular vectors describing the orientation of the sound source.

fDirXYZUpXYZ Pointer to an array of 6 floating point numbers.

Return Values

S_OK

E_INVALIDARG if NULL pointers are passed in.

Description

IA3dSource2::SetOrientation sets the orientation of the sound source in 3D space, relative to the current matrix in effect when **IA3dGeom2::BindSource** is called. **IA3dSource2::GetOrientation** returns the position set by **IA3dSource2::SetOrientation**.

This only has an effect with directional (cone) sources.

See Also

IA3dSource2::Set/GetPosition
IA3dSource2::Set/GetVelocity
IA3dGeom2::BindSource

IA3dSource2::Set/GetOrientationAngles

Sets and gets the orientation of the sound source.

Prototype

```
HRESULT SetOrientationAngles3f(
    A3DVAL          fHeading, A3DVAL  fPitch, A3DVAL  fRoll,
);

HRESULT SetOrientationAngles3fv(
    LPA3DVAL        fHPR,
);

HRESULT GetOrientationAngles3f(
    LPA3DVAL        pHeading, LPA3DVAL  fPitch, LPA3DVAL  fRoll,
);

HRESULT GetOrientationAngles3fv(
    LPA3DVAL        fHPR,
);
```

Parameters

fHeading, *fPitch*, *fRoll* Euler angles describing the orientation of the source.

fHPR Array of three A3DVALs describing heading, pitch and roll.

Return Values

S_OK

E_INVALIDARG if NULL pointers are passed in.

Description

IA3dSource2::SetOrientationAngles sets the orientation of the source in 3D space. The parameters it takes are rotation values in degrees. *fHeading* represents rotation around the Y (up) axis, *fPitch* rotation about the X (right) axis, and *fRoll* rotation about the Z (out) axis. The rotations are applied in the following order: *fHeading*, *fPitch*, and *fRoll*.

The rotation described by the three angles is relative to the transformation applied to the listener. If **IA3dGeom2::BindSource** is not being used, there is no transformation applied to the source so the angles are absolute. If **IA3dGeom2::BindSource** is being used, the rotation is relative to the coordinate system described by the current matrix when **IA3dGeom2::BindSource** was called.

Using this method to set the source orientation will override the results of using **IA3dSource2::SetOrientation**. The two methods simply use different inputs to perform the same function.

IA3dSource2::GetOrientationAngles returns the rotation angles of the source relative to its coordinate system. If **IA3dSource2::SetOrientation** has been used, the angles will have been computed from the two vectors supplied to that method, so those angles will be returned rather than the last set of angles sent to **IA3dSource2::SetOrientationAngles**. It's worth noting that the same orientation can be described by more than one set of rotation angles. For example, heading, pitch and roll of 0, 0, 0 is the same as 180, 180, 180. If the orientation was set using the vector method, **IA3dSource2::GetOrientationAngles** might not return the most obvious angles for that orientation although they will be correct.

See Also

IA3dSource2::SetOrientation
IA3dGeom2::BindSource

IA3dSource2::Set/GetPanValues

Sets the gains for multi-channel, non-spatialized sources.

Prototype

```
HRESULT SetPanValues(
    DWORD       dwNumValues,
    LPA3DVAL    fGains
);

HRESULT GetPanValues(
    DWORD       dwNumValues,
    LPA3DVAL    fGains
);
```

Parameters

dwNumValues Number of pan values being sent in the array.

fGains Gain values for each channel.

Return Values

S_OK
A3DERROR_SOURCE_IN_A3D_MODE
A3DERROR_INVALID_NUMBER_OF_CHANNELS

Description

By default, sources are spatialized in 3D and their gains are computed as factors of distance, location, and occlusion attenuation in addition to any gain value explicitly set by the application through **IA3dSource2::SetGain** and **IA3d5::SetOutputGain**. Using **IA3dSource2::SetRenderMode** to switch a source to native mode means that it is no longer subject to any 3D processing and enables this method to set its pan values explicitly.

Only two channel playback is currently supported in native mode, so only left and right gain values can be passed in to **IA3dSource2::SetPanValues**. As such, *dwNumValues* should be 2. The valid range for the gain values is the same as for all the other methods used to set gains - 0.0 to 1.0 inclusive.

This method will return an error if the source is not in native mode, but it will still set the pan values meaning that when the source is switched to native mode the values will be applied.

See Also

IA3dSource2::Set/GetRenderMode
IA3dSource2::Set/GetGain
IA3d5::Set/GetOutputGain

IA3dSource2::Set/GetPitch

Sets and gets the playback pitch bend of a source.

Prototype

```
HRESULT SetPitch(
    A3DVAL          fPitch
);

HRESULT GetPitch(
    LPA3DVAL        pfPitch
);
```

Parameters

pPitch Pitch bend factor.

Return Values

S_OK
E_INVALIDARG

Description

IA3dSource2::SetPitch lets an application change the playback rate of a source. A value of 2.0 shifts the source up an octave and a value of 0.5 shifts it down an octave. Valid ranges depend on the input sample rate of the sound source and how much Doppler is being applied, but in most cases 0.5 – 2.0 is valid. The default value is 1.0 meaning that the source's pitch is unaltered.

See Also

None.

IA3dSource2::Set/GetPlayPosition

Sets the playback cursor in a sound source to a particular time.

Prototype

HRESULT **SetPlayPosition**(
    DWORD *dwOffset*
);

Parameters

dwOffset Number of bytes from the beginning of the wave data.

Return Values

S_OK
E_INVALIDARG

Description

IA3dSource2::SetPlayPosition moves the playback cursor in the wave data for a sound source to a particular point, *dwOffset* bytes from the beginning of the wave. This position is sample accurate.

If a position greater than the length of the wave data is specified the method returns E_INVALIDARG. Calling this method does not affect the playback state of the sound source.

IA3dSource2::GetPlayPosition returns the position of the playback cursor in the wave data for the source.

See Also

IA3dSource2::Rewind
IA3dSource2::SetPlayTime

IA3dSource2::Set/GetPlayTime

Sets and gets the playback cursor in the wave data.

Prototype

```
HRESULT SetPlayTime(
    A3DVAL          fSeconds
);

HRESULT GetPlayTime(
    LPA3DVAL        fSeconds
);
```

Parameters

fSeconds	Floating point number specifying seconds from the start of the wave data.

Return Values

S_OK
E_INVALIDARG

Description

IA3dSource2::SetPlayTime moves the playback cursor in the wave data for a sound source to a particular point, *fTime* seconds from the beginning of the wave. This method provides the same functionality as **IA3dSource2::SetPlayPosition** with a different input parameter.

If a time greater than the length of the wave data is specified then method returns E_INVALIDARG. Calling this method does not affect the playback state of the sound source.

IA3dSource2::GetPlayTime returns the position of the playback cursor in the wave data for the source.

See Also

IA3dSource2::Set/GetPlayPosition

IA3dSource2::Set/GetPosition

Sets and gets the location of a sound source.

Prototype

```
HRESULT SetPosition3f(
    A3DVAL          x, A3DVAL y, A3DVAL z
);

HRESULT GetPosition3f(
    LPA3DVAL        px, LPA3DVAL py, LPA3DVAL pz
);

HRESULT SetPosition3fv(
    LPA3DVAL        vxyz
);

HRESULT GetPosition3fv(
    LPA3DVAL        vxyz
);
```

Parameters

x, *y*, *z*	Three floats for the position of the sound source.
vxyz	An array of three floats for the position of the sound source.

Return Values

S_OK

Description

IA3dSource2::SetPosition sets the location of the sound source in 3D space, relative to the current matrix in effect when **IA3dGeom2::BindSource** is called.
IA3dSource2::GetPosition returns the position set by **IA3dSource2::SetPosition**.

See Also

IA3dSource2::Set/GetOrientation
IA3dSource2::Set/GetVelocity
IA3dGeom2::BindSource

IA3dSource2::Set/GetPriority

Sets the priority of the source.

Prototype

HRESULT **SetPriority**(
    A3DVAL *fPriority*
);

HRESULT **GetPriority**(
    LPA3DVAL *pfPriority*
);

Parameters

fPriority A floating point number between 0.0 and 1.0 inclusive (default 0.5).

Return Values

S_OK
E_INVALIDARG

Description

IA3dSource2::SetPriority lets the application assign priorities to sound sources. The Resource Manager assigns a weight to each source based on a combination of priority and audibility. The bias of the weighting function can be globally modified with **IA3d5::SetRMPriorityBias**.

All sound sources have a default priority of 0.5 with lowest priority being 0.0 and highest 1.0.

See Also

IA3d5::SetRMPriorityBias

IA3dSource2::Set/GetReflectionDelayScale

Scales the reflection delays for a source.

Prototype

```
HRESULT SetReflectionDelayScale(
    A3DVAL          fScale
);

HRESULT GetReflectionDelayScale(
    LPA3DVAL        fScale
);
```

Parameters

fScale Non-negative floating point number specifying the delay scale.

Return Values

S_OK
E_INVALIDARG

Description

This method scales the delays of all the reflections generated by a source. It can be used to exaggerate the effect of reflections when *fScale* is greater than 1.0 (the default). *fScale* can be any positive number, but reflection delays are still clamped at the value set in **IA3d5::SetMaxReflectionDelayTime**, or the default of 0.3 seconds if that method wasn't called.

The delay scaling applied by this method is multiplied by the delay scaling set using **IA3dGeom2::SetReflectionDelayScale**. If either is set to 0.0, reflections will not be delayed at all. Setting either value to 0.0 is not recommended.

See Also

IA3dSource2::SetReflectionGainScale
IA3dGeom2::SetReflectionDelayScale
IA3dGeom2::SetReflectionGainScale

IA3dSource2::Set/GetReflectionGainScale

Scales the reflection gains for a source.

Prototype

```
HRESULT SetReflectionGainScale(
    A3DVAL          fScale
);

HRESULT GetReflectionGainScale(
    LPA3DVAL        fScale
);
```

Parameters

fScale Non-negative floating point number specifying the gain scale.

Return Values

S_OK
E_INVALIDARG

Description

This method scales the gains of all the reflections generated by a source. It can be used to exaggerate the effect of reflections when *fScale* is greater than 1.0 (the default). *fScale* can be any positive number, but reflection gains are clamped at 1.0.

The gain scaling applied by this method is multiplied by the gain scaling set using **IA3dGeom2::SetReflectionGainScale**. If either is set to 0.0, reflections will be silent.

See Also

IA3dSource2::SetReflectionDelayScale
IA3dGeom2::SetReflectionDelayScale
IA3dGeom2::SetReflectionGainScale

IA3dSource2::Set/GetRenderMode

Controls how a source is rendered.

Prototype

```
HRESULT SetRenderMode(
    DWORD       dwMode
);

HRESULT GetRenderMode(
    LPDWORD     pdwMode
);
```

Parameters

dwMode Mode bit mask for source rendering. Specify:

A3DSOURCE_RENDERMODE_A3D
A3DSOURCE_RENDERMODE_MONO
A3DSOURCE_RENDERMODE_1ST_REFLECTIONS
A3DSOURCE_RENDERMODE_OCCLUSIONS
A3DSOURCE_RENDERMODE_NATIVE
A3DSOURCE_RENDERMODE_DEFAULT

Return Values

S_OK
E_FAIL

Description

This method allows the application to change the type and level of processing performed on a source. By default, sources are spatialized and will render reflections and occlusions if geometry is being used. This default mode is equivalent to *dwMode* = A3DSOURCE_RENDERMODE_A3D | A3DSOURCE_RENDERMODE_OCCLUSIONS | A3DSOURCE_RENDERMODE_1ST_REFLECTIONS.

Switching the render mode to A3DSOURCE_RENDERMODE_NATIVE will disable all 3D processing and play the source back in its native format. This enables **IA3dSource2::SetPanValues**, allowing the source to be panned between two output channels. In this mode, calling any of the 3D or geometry methods for a source will return an error, though the property will still be set and applied when the source is switched to A3D mode.

A3DSOURCE_RENDERMODE_MONO leaves the source as A3D but bypasses the HRTF and distance model processing. Again, geometry has no effect on sources in this mode.

Sources that were created as unmanaged can't be switched between A3D and native modes, though unmanaged A3D sources can still have reflections and occlusions and mono toggled on and off.

The A3D, mono and native modes are mutually exclusive.

See Also

IA3d5::NewSource
IA3dSource2::Set/GetPanValues

IA3dSource2::Set/GetReverbMix

Sets or gets the mix level in the current reverb for this source.

Prototype

```
HRESULT SetReverbMix(
    A3DVAL Level,
    A3DVAL HiFreq
);

HRESULT GetReverbMix(
    A3DVAL *pLevel,
    A3DVAL *pHiFreq
);
```

Parameters

Level	Wet/dry reverberation amount for this source.
HiFreq	Low-pass filter amount for this source.
pLevel	Pointer to be filled with the wet/dry reverberation amount.
pHiFreq	Pointer to be filled with low-pass filter amount.

Return Values

S_OK
A3DERROR_INVALID_ARGUMENT
A3DERROR_FEATURE_NOT_REQUESTED

Description

The SetReverbMix function sets the mix level in the reverb for this source

The Level parameter controls the wet/dry mix of this source in the current reverb. This value is a percentage, and can range from 0.0 (completely dry) to 1.0 (completely wet).

The HiFreq parameter is a relative adjustment to the direct path's low-pass filter. This value is a percentage, and can range from 0.0 (no low-pass effect) to 1.0 (full low-pass effect). The low-pass filter affects the early reflections and reverb identically by reducing their energy at high frequencies.

For these functions to succeed, reverb must have been requested in the **IA3d5::InitEx** call by including the A3D_REVERB flag. If this was not the case, A3DERROR_FEATURE_NOT_REQUESTED will be returned. In addition, the

source’s render mode must be A3D. If not, calling SetReverbMix will return the error code A3DERROR_SOURCE_IN_NATIVE_MODE, though the properties will still be set and applied when the source is switched to A3D mode.

See Also

IA3d5::InitEx

IA3dReverb

IA3dSource2::Set/GetTransformMode

Sets the transform mode for a source.

Prototype

```
HRESULT SetTransformMode(
    DWORD    dwMode
);

HRESULT GetTransformMode(
    LPDWORD dwMode
);
```

Parameters

dwMode	Transform mode to be used. Specify one of:
	A3DSOURCE_TRANSFORMMODE_NORMAL
	A3DSOURCE_TRANSFORMMODE_HEADRELATIVE

Return Values

S_OK
E_INVALIDARG
A3DERROR_SOURCE_IN_NATIVE_MODE

Description

A source has properties to specify its position, orientation and velocity in the world. Sometimes it can be useful to redefine the origin for the source, making those properties relative to a location other than the origin of the world. This can be achieved in two ways. One is to use **IA3dGeom2::BindSource**, which applies the current matrix to the source meaning that its position, orientation and velocity will be transformed by that matrix. This allows sources to be easily attached to objects which are moving around in the world, but requires that the application be using the **IA3dGeom2** interface. If the requirement is simply to attach a source to the listener, setting the source into listener-relative coordinates using **IA3dSource2::SetTransformMode** is easier than using the matrix stack and binding the source to the same matrix as the listener.

Possible values for *dwMode* are A3DSOURCE_TRANSFORMMODE_NORMAL and A3DSOURCE_TRANSFORMMODE_HEADRELATIVE. By default, sources are set to the former.

When a source is in head relative mode, all tranformations of that source are relative to the listener. If **IA3dGeom2::BindSource** was used to apply a tranformation matrix to

the source, that tranformation is applied after transforming to listener coordinates. Even binding the source to an identity matrix will not locate the source at the origin of the world if it is in head relative mode.

This method sets a mode for the source that stays in effect until the method is called again to change it. It is not necessary to call this method with the same parameter every frame.

See Also

IA3dGeom2::BindSource
IA3dGeom2::BindListener

IA3dSource2::Set/GetVelocity

Sets and gets the velocity of a source.

Prototype

HRESULT **SetVelocity3f**(
    A3DVAL *vx*, A3DVAL *vy*, A3DVAL *vz*
);

HRESULT **SetVelocity3fv**(
    LPA3DVAL *vxyz*
);

HRESULT **GetVelocity3f**(
    LPA3DVAL *pvx*, LPA3DVAL *pvy*, LPA3DVAL *pvz*
);

HRESULT **GetVelocity3fv**(
    LPA3DVAL *vxyz*
);

Parameters

vx, *vy*, *vz*	Three floats for the velocity vector of the source.
vxyz	An array of three floats for the velocity vector of the source.

Return Values

S_OK

Description

IA3dSource2::SetVelocity sets the velocity vector of a source relative to the current matrix in effect when **IA3dGeom2::BindSource** is called. This information is used to compute Doppler shift. If the source isn't bound to a matrix the velocity vector passed in to this method represents the absolute velocity of the source in the world.

See Also

IA3dSource2::Set/GetPosition
IA3dSource2::Set/GetOrientation
IA3dGeom2::BindSource

IA3dSource2::Set/GetVolumetricBounds

Prototype

HRESULT **SetVolumetricBounds**(
A3DVAL *dx*,
A3DVAL *dy*,
A3DVAL *dz*
);

HRESULT **GetVolumetricBounds**(
LPA3DVAL *pdx*,
LPA3DVAL *pdy*,
LPA3DVAL *pdz*
);

Parameters

dx, *dy*, *dz* — dimensions of the bounding box, in full length. dx = width, dy = height, dz = depth.

pdx, *pdy*, *pdz* — return pointer values to the dimensions of the bound box.

Return Values

E_INVALIDARG
E_POINTER
S_OK

Description

By default, a source is considered a point source and has volumetric dimensions (dx, dy, dz) all set to 0. If any one of the dimension values is given a value greater than zero, the source will be rendered with volumetric properties.

The orientation of the boundaries is controlled by the source orientation functions **IA3dSource2::Get/Set Orientation**, **IA3dSource2::Get/SetOrientationAngles**, and **IA3dGeom2::BindSource**.

Volumetric source are rendered differently than point sources. Their distance gain, heading, and occlusion properties are treated different and depends on the dimensional size of the volume of the source.

The rendering characteristics of a volumetric source can be changed using **IA3dSource2::Set/GetVolumetricDamping**.

When setting the bounds dimension only valid values will be accepted. Invalid values will be ignored and an error E_INVALIDARG will be returned.

See Also:

IA3dSource2::Set/GetVolumetricDamping

IA3dSource2::Set/GetVolumetricDamping

Sets and gets the damping factors used to rendering the volumetric source.

Prototype

HRESULT **SetVolumetricDamping**(
    A3DVOLSRCDAMPINFO **pVolSrcDampInfo*
);

HRESULT **GetVolumetricDamping**(
    A3DVOLSRCDAMPINFO **pVolSrcDampInfo*
);

Parameters

pVolSrcDampInfo Pointer to the structure damp info structure (described below)

Return Values

E_INVALIDARG
E_POINTER
S_OK

Description

This function allows tweaking of the rendering characteristics of a volumetric source. It only applies to sources that are rendered as a volumetric source, and these are defined as sources that have non-zero dimensional bounds.

When determining the amount a volumetric source is occluded, we use two values: size-relative fraction and a visibility-relative fraction.

The size-relative damping fraction approximates the percentage of the source that is occluded by the polygon. It is defined as:

```
(size of occluding polygon) /
((size of occluding polygon) + (size of the sound source))
```

The visibility-relative damping fraction is the number of corner points that are occluded. It is defined as:

```
(number of corner points occluded) / (nTestPointsMax * 2)
```

The only parameter is a pointer to an A3DVOLSRCDAMPINFO structure, which contains information that will affect the rendering of a volumetric Source.

When setting the structure only valid values will be accepted. Invalid values will be ignored and an error E_INVALIDARG will be returned.

See Also

A3DVOLSRCDAMPINFO

IA3dSource2::Set/GetVolumetricBounds

IA3dSource2::SetPlayEvent

Sets an event to be triggered at a certain point in the wave data.

Prototype

```
HRESULT SetPlayEvent(
    DWORD   dwOffset,
    HANDLE  hEvent
);
```

Parameters

dwOffset	Offset in bytes from beginning of wave data at which to trigger the event.
hEvent	Handle to an event.

Return Values

S_OK
E_OUTOFMEMORY
A3DERROR_NO_WAVE_DATA
A3DERROR_FAILED_QUERY_DIRECTSOUNDNOTIFY
A3DERROR_FAILED_DIRECTSOUNDNOTIFY

Description

This method makes it possible to place markers in the wave data of a source and have A3D trigger events when the playback cursor reaches each of the markers. **IA3dSource2::SetPlayEvent** places a single marker with its associated event in the wave data, and it can be called any number of times to add multiple markers. Calling it twice with the same *dwOffset* value overwrites the previous event with the new one. Setting the event to NULL clears the event previously set for that location. *hEvent* is a Windows event handle created with the Win32 API call **CreateEvent**.

There is a special case value for *dwOffset*, A3DSOURCE_WAVEEVENT_STOP, which triggers the event when the source is stopped either because the application called **IA3dSource2::Stop** or because the end of the data was reached in a non-looping source.

Also note the following:

Event positions are in bytes and values equal to or greater than the allocated wave data size are not signaled.

A source's position macro A3DSOURCE_WAVEEVENT_STOP will not signal if the source is played as A3D_LOOPED.

Duplicated sources do not inherit the Event triggers placed on the original source.

A3D is not thread safe. When processing the events in a different thread, use some interlock scheme (such as Mutex or Semaphores) with the main thread when operating on any A3D objects.

See Also

IA3dSource2::ClearPlayEvents
IA3dSource2::Stop

IA3dSource2::Stop

Stops a source playing.

Prototype

```
HRESULT Stop(
    void
);
```

Parameters

None.

Return Values

S_OK
A3DERROR_NO_WAVE_DATA
A3DERROR_FAILED_STOP

Description

IA3dSource2::Stop stops a sound source playing. It sends a signal to the resource manager to tell it to remove this sound source from its play list. Unlike **IA3dSource2::Play**, this method is applied immediately and is not deferred until **IA3d5::Flush** is called.

See Also

IA3dSource2::Stop

IA3dSource2::Unlock

Unlocks a previously locked sound source.

Prototype

```
HRESULT Unlock(
    LPVOID    pvAudioPtr1,
    DWORD     dwNumBytes1,
    LPVOID    pvAudioPtr2,
    DWORD     dwAudioBytes2
);
```

Parameters

pvAudioPtr1 Address of the value retrieved from **IA3dSource2::Lock**.

dwNumBytes1 Number of bytes written to the first block.

pvAudioPtr2 Address of the value retrieved from **IA3dSource2::Lock**.

dwNumBytes2 Number of bytes written to the second block.

Return Values

S_OK
A3DERROR_NO_WAVE_DATA
A3DERROR_FAILED_UNLOCK_BUFFER

Description

Following a call to **IA3dSource2::Lock** and data being copied to the sound source, **IA3dSource2::Unlock** enables the new data to be played back.

See Also

IA3dSource2::Lock
IA3dSource2::AllocateAudioData
IA3dSource2::LoadFile

A3DCAPS_SOURCE

Description

This structure is used in **IA3dSource2::GetInfo**, and wraps the information structures used for the various audio formats supported by A3D 3.0. Currently, these consist of PCM Wave, MP3, and AC-3 files.

Members

DWORD *dwSize*	Always set this as sizeof(A3DCAPS_SOURCE).
DWORD *dwType*	Specifies the structure which the Union pointer of this A3DCAPS_SOURCE points to. Valid values are: **A3DSOURCE_FORMAT_WAVE**, **A3DSOURCE_FORMAT_MP3**, and **A3DSOURCE_FORMAT_AC3.**
char **szFilename*	This pointer is filled in with the name of the file that the current source is playing, if it has a file. This value is temporary and is only valid in the scope of the calling function and while the source is still valid..
union *data*	Union of the following structures:

A3DSOURCE_WAVEFORMAT *data.waveFormat*

Holds data for a PCM-WAV source. This pointer is only valid if dwType is **A3DSOURCE_FORMAT_WAVE.**

A3DSOURCE_MP3INFO *data.mp3Info*

Holds data for a MP3 source. This pointer is only valid if dwType is **A3DSOURCE_FORMAT_MP3.**

A3DSOURCE_AC3INFO *data.ac3Info*

Holds data for an AC-3 source. This pointer is only valid if dwType is **A3DSOURCE_FORMAT_AC3.**

A3DSOURCE_MP3INFO

Description

This structure contains information about MP3 audio data, and is only used in with sources that are playing MP3 data. This is encoded information, and isn't necessarily the number of channels or the sample rate the output buffer is playing at.
IA3dSource2::GetAudioFormat gives you that information.

Members

INT *nMpegLayer*	Major version number.
INT *nMpegVersion*	Minor version number.
INT *nBitrate*	Encoded bitrate.
INT *nChannels*	Number of channels in the decoded stream – usually 1 or 2.
INT *nSamplerate*	Sample rate of the decoded samples.
INT *nBitsPerSample*	Number of bits per decoded sample.
FLOAT *fTotalPlayLength*	Time of mp3 playback in seconds.

A3DSOURCE_WAVEFORMAT

Description

This structure is used to describe a chunk or stream of PCM-wave audio data. It is identical to the Windows WAVEFORMATEX structure, and has the same purpose.

Members

DWORD *dwSize*	Size of the A3DSOURCE_WAVEFORMAT structure. Must be set.
WORD *nChannels*	Number of discreet audio channels in this stream. Mono is 1, stereo is 2, quadraphonic is 4.
DWORD *nSamplesPerSec*	Number of samples to be played per second. Standard values include 11025, 22050, 44100, 48000.
DWORD *nAvgBytesPerSec*	Number of bytes to be played per second. Defined as **nChannels * wBitsPerSample * nSamplesPerSec**.
WORD *nBlockAlign*	Same as sample size. Defined as **nChannels * wBitsPerSample**.
WORD *wBitsPerSample*	Size of each individual sample. Usually 8 or 16 bits.

A3DVOLSRCDAMPINFO

Description

The A3DVOLSRCDAMPINFO structure specifies an application specified reverb. The descriptions of the various values are below.

Members

DWORD *dwSize*	The size of this structure. Always set the dwSize member before using the Set/Get functions.
A3DVAL *fAzimuthPan*	This number exaggerates the effect. Setting it to 0 completely eliminates the volumetric effect, while setting it to 1.0 takes it to extreme, so that the source behaves as if every point on the surface of the volume generated audio at full volume. The default value is 0.5f.
A3DVAL *fDampWeighting*	Indicates the weighting to be applied to the size-relative and visibility-relative damping fractions. A value of 0 gives all the weighting to the size-relative fraction, and none to the visibility-relative fraction. A value of 1 gives all the weighting to the visibility-relative fraction. The default value is 0.7f, giving 70% weighting to the visibility-relative fraction, and 30% to the size-relative fraction.
A3DVAL *fSizeDampMin*	This defines the minimum value the **size-relative fraction** can be. Setting it to 0.0 allows the source to be minimally occluded, while setting it to 1.0 means that any occluding polygon, regardless of its size, will fully occlude this source. By default the value is 0.5f. This fraction is balanced against the visibility-relative damping fraction and used in the final occlusion damping calculation as described below. The default value is 0.5f. Note that the distances from the listener to the occluding polygon and to the source are not considered by this fraction.
INT *nTestPointsMax*	Indicates the number of test points to use for the visibility-relative damping fraction. The occlusion algorithm casts a ray from each test point to the listener

	and checks the ray for intersection with the occluding polygon. It then calculates the visibility-relative damping fraction. This fraction is balanced against the size-relative damping fraction and used in the final occlusion damping calculation as described above. The minimum value is zero, which results in a value of 1 for the visibility-relative damping fraction. The maximum value is 6, which results in the testing of all points on the surface of the sound source. The default value is 6. In general, this value should be left at 6. It's useful only in edge cases, such as optimizing the rendering of hundreds of volumetric sources or thousands of occlusions polygons.
BOOL *bMonoInside*	This value determines how the source is rendered when the listener is inside the volume. Setting this value to TRUE renders the Source as mono, so it plays at full volume from all four speakers. Setting this to false renders the source as a point source. The default is FALSE.

Chapter 5: Property Set Reference Pages

IA3dPropertySet Interface

The A3D API supports property sets through the interface **IA3dPropertySet** that is similar to the DirectSound **IKsPropertySet**.

```
DECLARE_INTERFACE_(IA3dPropertySet, IUnknown)
{
    // IUnknown Methods.
    STDMETHOD(QueryInterface)
    STDMETHOD_(ULONG, AddRef)
    STDMETHOD_(ULONG, Release)

    // IA3dPropertySet Methods.
    STDMETHOD(QuerySupport)
    STDMETHOD(Get)
    STDMETHOD(Set)
    STDMETHOD(AddInitialStateParameters)
};
```

Methods IA3dPropertySet::QuerySupport, IA3dPropertySet::Get, and IA3dPropertySet::Set are the same as documented in DirectSound IKsPropertySet.

The function, **IA3dPropertySet::AddInitialStateParameter** is a required function that sets the "Zero" state of a property set. This is a required function because A3D sources can be resource managed. A zero state of the property set is needed before property sets can work.

In A3D, there are two types of property sets. A global property set and a buffer specific property set. The following table lists the differences of the global and buffer specific property sets.

	Global	**Buffer specific**
Getting the Interface	Query from the root **IA3Dx** interface.	Query from the **IA3dSource2** interface, AFTER the source has allocated wave data.
Purpose	For global application of property sets.	For buffer specific application of property sets.
Timing Effect	Applied immediately.	Only applied if buffer is in hardware.
IA3dPropertySet::AddInitalStateParam Function	Works as described.	Not applicable.

Sample Code

The code flows like this:

```
// Query for global property set.interface
IA3dPropertySet *pGlobalPropertySet = NULL;
g_pA3d->QueryInterface(IID_IAdPropertySet, &pGlobalPropSet);

// Query for Support
ULONG ulSupport;
ULONG ulID = 0;
pGlobalPropSet->QuerySupport(IID_CardSpecificGuid, ulItem, &ulSupport);

// Call AddInitialStateParameter to set the Zero state.
if (ulSupport &(KSPROPERTY_SUPPORT_GET|KSPROPERTY_SUPPORT_SET) == FALSE)
      return E_FAIL;

PROPERTYSETDATA PropSetData;
ZeroMemory(&PropSetData, sizeof(PROPERTYSETDATA);
pGlobalPropSet->AddInitialStateParameters(IID_CardSpecificGuid,
                                          ulID,&PropSetInstance,
                                          sizeof(PropSetInstance),
                                          &PropSetData,
                                          sizeof(PropSetData));

// You can release the global Interface now
// if you have no more global settings.
pGlobalPropertySet->Release();

// New a source, Load data
g_pA3d->NewSource(&pSource);
pSource->LoadWaveData("heli.wav");

// Query for buffer, property set.
IA3dPropertySet *pBufferPropSet=NULL;
pSource->QueryInterface(IID_IA3dPropertySet, &pGlobalPropSet);

// Set/Get property set on buffer.
...
```

The sample program propsets.cpp will demonstrate the use of property sets through the A3D API.

You will need a sound card that supports property sets for the program to work.

IA3dPropertySet::AddInitialStateParameters

This method sets the "zero" state of a property. This is a required function, needed before property sets can be used. Is valid only in the global context.

Prototype

HRESULT **AddInitialStateParameter**(
    REFGUID *rguidPropSet*,
    ULONG *ulId*,
    LPVOID *pInstanceData*,
    ULONG *ulInstanceLength*,
    LPVOID *pPropertyData*,
    ULONG *ulDataLength*
);

Parameters

rguidPropSet	Reference to (C++) or address of (C) a GUID representing the property set to be accessed.
ulId	Item within the property set to be accessed. Items are indexed from 0 and are always the same for a given property set.
pInstanceData	Instance data for the set call. If there are multiple objects within the port that this operation can act on, the instance data specifies which object should be used. No standard property set items at the DirectMusic API level use instance data; however, vendor-defined extensions are free to use it.
ulInstanceLength	Number of bytes pointed to by *pInstanceData*
pPropertyData	Property data to set for this item.
ulDataLength	Number of bytes pointed to by *pPropertyData*

Return Values

S_OK	The method succeeded.
E_INVALIDARG	One of the arguments was passed an invalid value.

E_POINTER	One of the arguments is a NULL pointer and a non-NULL pointer is expected.
E_OUTOFMEMORY	Could not allocate memory for the *CPropertySetItem*.

Description

This method must be called at least once before the **IA3dPropertySet::Set** method may be called. When streaming buffers are swapped around we need to restore the state of the hardware buffer to some sort of initial state so we don't inherit the last buffers properties into the new one.

See Also

None.

IA3dPropertySet::AddRef

Increments the **IA3dPropertySet** reference count.

Prototype

ULONG **AddRef**(
    void
);

Parameters

None.

Return Values

Returns the new reference count.

Description

When going through a COM method such as **QueryInterface** or **NewSource** to get an interface pointer to a component, the reference count of the component is automatically incremented. The reference count is used to let the component know when nothing is accessing it anymore and that it can delete itself from memory.

Whenever an interface pointer is assigned to another interface pointer, the **AddRef** method should be called to let the component know that two pointers are using the same interface. Now when the **Release** method is called, the component won't delete itself since it has been told something else is still using it. Consider the following example:

```
hr = pRoot->QueryInterface(IID_IBox, (void **)&pBox1);
if (SUCCEEDED(hr))
{
   pBox1->DrawIt();
   pBox2 = pBox1;
   pBox2->AddRef();
   pBox1->Release();
}
```

While *pBox1* is now invalid because it has been released, *pBox2* remains intact and can still be used.

All A3D 3.0 interfaces inherit the COM **IUnknown** interface that contains the methods **AddRef**, **QueryInterface**, and **Release**. *Inside COM* by Microsoft Press is an excellent resource for detailed information on COM.

See Also

IA3dPropertySet::QueryInterface
IA3dPropertySet::Release

IA3dPropertySet::Get

The **IKsPropertySet::Get** method retrieves data for an item in a property set.

Prototype

```
HRESULT Get(
    REFGUID  rguidPropSet,
    ULONG    ulId,
    LPVOID   pInstanceData,
    ULONG    ulInstanceLength,
    LPVOID   pPropertyData,
    ULONG    ulDataLength,
    ULONG    *pulBytesReturned
);
```

Parameters

rguidPropSet	Reference to (C++) or address of (C) a GUID representing the property set to be accessed.
ulId	Item within the property set to be accessed. Items are indexed from 0 and are always the same for a given property set GUID.
pInstanceData	Instance data for the get call.
ulInstanceLength	Number of bytes pointed to by *pInstanceData*.
pPropertyData	Data to set for this item.
ulDataLength	Number of bytes pointed to by *pPropertyData*
pulBytesReturned	Address of a variable to receive the number of bytes written into *pPropertyData*.

Return Values

Return values are determined by the designer of the property set.

If the method succeeds, the return value may be S_OK.

If it fails, the method may return E_POINTER.

Description

The format of the data in both *pInstanceData* and *pPropertyData* is item-specific.

See Also

None.

IA3dPropertySet::QueryInterface

Returns an interface pointer for a supported interface.

Prototype

```
HRESULT QueryInterface(
    REFIID    iid,
    LPVOID    *pInterface
);
```

Parameters

iid	Interface identifier. Specify only IID_IA3dPropertySet.
pInterface	Address of a pointer to an interface which will be filled out by the method

Return Values

S_OK
E_NOINTERFACE

Description

All A3D interfaces inherit the **IUnknown** interface that contains a method called **QueryInterface**. This method is used to let the application know what other interfaces a particular interface supports, and to return a pointer to a requested interface if it is supported. The different A3D interfaces support different interfaces.

The **IA3dPropertySet** interface doesn't support any other interfaces, so the only valid value for *iid* is IID_IA3dPropertySet, which will return another listener interface pointer and increment the reference count.

Calling any **QueryInterface** and asking for an interface that isn't supported will return the error E_NOINTERFACE. The address of the pointer passed in to the method will be left at the value it was set to by the calling method, so it may not be NULL. For this reason, it is essential to check the return value of this method.

See Also

IA3dPropertySet::AddRef
IA3dPropertySet::Release

IA3dPropertySet::QuerySupport

Determines whether a property in a property set is supported on the port or device.

Prototype

```
HRESULT QuerySupport(
    REFGUID  rguidPropSet,
    ULONG    ulId,
    ULONG    *pulTypeSupport
);
```

Parameters

rguidPropSet Reference to (C++) or address of (C) a GUID representing the property set to be queried.

ulId Item within the property set to be accessed. Items are indexed from 0 and are always the same for a given property set.

pulTypeSupport Address of a variable to receive information about support for the property. If the property or property set does not exist, this is set to 0. Otherwise it may contain one or both of the following flags:

KSPROPERTY_SUPPORT_ GET	The property item may be retrieved.
KSPROPERTY_SUPPORT_SET	The property item may be set.

Return Values

Return values are determined by the designer of the property set.

If the method succeeds, the return value may be S_OK. (See Description.)

If it fails, the method may return one of the following error values:

E_NOTIMPL (See Description.)

E_POINTER

Description

Whether it is valid to support some properties within the set but not others depends on the definition of the property set. Consult the hardware manufacturer's specification for the property set of interest.

Some implementations may return S_OK when the property is not supported, and others may return E_NOTIMPL. To be sure that a property is supported you should check both the return value of the method and the value returned in *pulTypeSupport*.

See Also

None.

IA3dPropertySet::Release

Decrements the IA3dPropertySet reference count.

Prototype

```
ULONG Release(
    void
);
```

Parameters

None.

Return Values

Returns the new reference count.

Description

When going through a COM method such as **QueryInterface** or **NewSource** to get an interface pointer to a component, the reference count of the component is automatically incremented. The reference count is used to let the component know when nothing is accessing it anymore and that it can delete itself from memory.

Calling **IA3dPropertySet::Release** decrements the reference count for the **IA3dPropertySet** interface, and if it is 0, the object deletes itself from memory.

All A3D 3.0 interfaces inherit the COM **IUnknown** interface that contains the methods **AddRef**, **QueryInterface**, and **Release**. *Inside COM* by Microsoft Press is an excellent resource for detailed information on COM.

See Also

IA3dPropertySet::AddRef
IA3dPropertySet::QueryInterface

IA3dPropertySet::Set

Method sets the value of a property in a property set.

Prototype

```
HRESULT Set(
    REFGUID rguidPropSet,
    ULONG   ulId,
    LPVOID  pInstanceData,
    ULONG   ulInstanceLength,
    LPVOID  pPropertyData,
    ULONG   ulDataLength
);
```

Parameters

rguidPropSet	Reference to (C++) or address of (C) a GUID representing the property set to be accessed.
ulId	Item within the property set to be accessed. Items are indexed from 0 and are always the same for a given property set.
pInstanceData	Instance data for the set call. If there are multiple objects within the port that this operation could act on, the instance data specifies which object should be used. No standard property set items at the DirectMusic API level use instance data; however, vendor-defined extensions are free to use it.
ulInstanceLength	Number of bytes pointed to by *pInstanceData*.
pPropertyData	Property data to set for this item.
UlDataLength	Number of bytes pointed to by *pPropertyData*.

Return Values

Return values are determined by the designer of the property set.

If the method succeeds, the return value may be S_OK.

If it fails, the method may return E_POINTER.

Description

The format of the data in both *pInstanceData* and *pPropertyData* is item-specific.

See Also

None.

Chapter 6: Geometry Engine Reference Pages

Aureal Wavetracing allows complex acoustic environments to be rendered in real time. Acoustic Environments are composed of surfaces made out of three- and four-sided polygons and the geometry engine renders the effect those polygons have on sound sources.

There are 3 interfaces within the geometry engine:

- IA3dGeom2
- IA3dList
- IA3dMaterial

IA3dGeom2 contains all the methods needed to manipulate the matrix stack, render polygons, and set different rendering modes. **IA3dList** allows sequences of calls to **IA3dGeom2** methods to be recorded and rendered at a later stage by calling a single method. **IA3dMaterial** allows the properties of an acoustic material to be defined. For more detail, refer to the introduction for each of the Geometry Engine interfaces.

IA3dGeom2 Interface

An acoustic scene consists of a collection of polygons that interact with sounds in the scene. The **IA3dGeom2** interface provides methods for:

- Describing the collection of polygons.
- Applying geometric transformations to the listener and sound sources.

To obtain the geometry interface, call the **QueryInterface** method of the existing interface on the **IA3d5** object:

```
IA3dGeom2 *pIA3dGeom2;
   pIA3d5->QueryInterface(IID_IA3dGeom2, (void **)&pIA3dGeom2);
```

If you're familiar with 3D graphics, think of an audio frame as being the same as a graphics frame. A frame starts by clearing the audio frame buffer with a call to the method **IA3d5::Clear**. A frame ends by refreshing the scene with a call to the method **IA3d5::Flush**. The geometry buffer is refilled for each frame — you can't just send the data that changed from the last frame since the scene is completely refreshed after each call to **IA3d5::Flush**. For example, if the scene contains a cube, the six polygons for the cube are sent to the geometry engine every frame between the calls to **IA3d5::Clear** and **IA3d5::Flush**. Polygons sent outside of these two calls are not rendered.

When the geometry engine is being used, **IA3d5::Flush** computes reflections and occlusions for the sounds in the scene. Reflections and occlusions are only computed correctly once the entire scene is described — this is why they are computed at the *end* of the frame with the call to **IA3d5::Flush**.

To add hierarchy to your scene, use matrices. A matrix is used to convert a point in 3D space from one coordinate system to another. The application doesn't need to bind the listener and sources to a matrix. By default, they're bound to an identity matrix so their positions and orientations are left according to how they are defined through the **IA3dSource2** and **IA3dListener** interfaces.

In addition to describing polygons and applying geometric transformations to the listener and source, the application can also:

- Define the acoustic properties of a surface by applying a material. Refer to “IA3dMaterial Interface” on page 192 for more information.
- Reduce computation in a scene by caching the geometry. Refer to “IA3dList Interface” on page 182 for more information.

IA3dGeom2::AddRef

Increments the IA3dGeom2 reference count.

Prototype

ULONG **AddRef**(
    void
);

Parameters

None.

Return Values

Returns the new reference count.

Description

When going through a COM method such as **QueryInterface** or **NewSource** to get an interface pointer to a component, the reference count of the component is automatically incremented. The reference count is used to let the component know when nothing is accessing it anymore and that it can delete itself from memory.

Whenever an interface pointer is assigned to another interface pointer, the **AddRef** method should be called to let the component know that two pointers are using the same interface. Now when the **Release** method is called, the component won't delete itself since it has been told something else is still using it. Consider the following example:

```
hr = pRoot->QueryInterface(IID_IBox, (void **)&pBox1);
if (SUCCEEDED(hr))
{
   pBox1->DrawIt();
   pBox2 = pBox1;
   pBox2->AddRef();
   pBox1->Release();
}
```

While *pBox1* is now invalid because it has been released, *pBox2* remains intact and can still be used.

All A3D 3.0 interfaces inherit the COM **IUnknown** interface that contains the methods **AddRef**, **QueryInterface**, and **Release**. *Inside COM* by Microsoft Press is an excellent resource for detailed information on COM.

See Also

IA3dGeom2::QueryInterface
IA3dGeom2::Release

IA3dGeom2::Begin, IA3dGeom2::End

Delimits the vertices of a primitive or a group like of primitives.

Prototype

```
HRESULT Begin(
    DWORD    dwType
);

HRESULT End(
    void
);
```

Parameters

dwType Type of primitive to build. Specify one of:

A3D_TRIANGLES
A3D_QUADS
A3D_SUB_TRIANGLES
A3D_SUB_QUADS

Return Values

S_OK
E_INVALIDARG

Description

Primitives describe the acoustic scene being rendered, and may reflect and occlude sound sources in that scene. **IA3dGeom2::Begin** and **IA3dGeom2::End** enclose calls to methods which define primitives. Between those calls, **IA3dGeom2::Normal** and **IA3dGeom2::Vertex** specify the normal vectors and vertex locations of the primitives. The *dwType* parameter specifies the type of primitive to construct. Three-dimensional primitives must be convex and coplanar.

Valid primitive types are A3D_TRIANGLES and A3D_QUADS for 3 and 4 sided polygons, and A3D_SUB_TRIANGLES and A3D_SUB_QUADS for 3 and 4 sided subfaces. Subfaces are polygons that are placed onto a parent polygon, allowing regions of a large surface to have different acoustic properties without splitting the parent polygon into a lot of smaller polygons. Subfaces have an opening factor applied to them, with 0.0 meaning the subface has no effect on the parent polygon and 1.0 meaning the area it covers is completely transparent to audio, as if there was no polygon covering that

area at all. This allows doors to be easily rendered while keeping the overall polygon count low.

Multiple primitives of the same type can be defined between a single **IA3dGeom2::Begin** and **IA3dGeom2::End** pair. A3D automatically ends each polygon when the correct number of vertices has been received and starts creating a new one if more data is sent. When the last vertex of a polygon is received, a normal will automatically be computed for the surface if **IA3dGeom2::Normal** wasn't used to specify one. When sending multiple polygons inside a single begin/end block, each group of vertices representing a polygon still has to be tagged with a unique ID, (by using **IA3dGeom2::Tag**), if reflections are being used.

The polygons created inside a begin/end block inherit the acoustic properties of the current material set by **IA3dGeom2::BindMaterial**. While it is possible to change the material before each vertex is sent, the current material in effect when the last vertex of a polygon is sent is the one that's applied to the entire polygon.

Only a subset of IA3dGeom2 methods can be used between **IA3dGeom2::Begin** and **IA3dGeom2::End**. Valid methods are IA3dGeom2::Normal, IA3dGeom2::Vertex, **IA3dGeom2::BindMaterial**, **IA3dGeom2::Tag**, **IA3dGeom2::SetOpeningFactor**, **IA3dGeom2::SetRenderMode**, and all **IA3dGeom2::Get** methods.

See Also

IA3dGeom2::Normal
IA3dGeom2::Vertex
IA3dGeom2::SetOpeningFactor
IA3dGeom2::Tag

IA3dGeom2::BindListener

Inserts the listener into the scene hierarchy.

Prototype

HRESULT **BindListener**(
    void
);

Parameters

None.

Return Values

S_OK

Description

IA3dGeom2::BindListener applies the current matrix to the listener and transforms its position, orientation and velocity. By default the listener has no transformations associated with it so its properties are in absolute world space. When the listener is bound to a matrix other than the identity matrix, its properties are in the coordinate system defined by that matrix.

This method is useful when attaching the listener to moving geometry. Rather than continually computing the world coordinates of the vertices for the object in which the listener is traveling, the object could be specified in local coordinates and moved by using **IA3dGeom2::Translate** and **IA3dGeom2::Rotate**. Binding the listener to the matrix that is applied to the geometry will cause it to move with the object without the application having to calculate its world location and orientation.

If **IA3dGeom2::BindListener** is being used, it should be called every frame to set the listener matrix up.

See Also

IA3dGeom2::BindSource
IA3dGeom2::PushMatrix
IA3dGeom2::PopMatrix
IA3dGeom2::Translate
IA3dGeom2::Rotate
IA3dGeom2::Scale
IA3dGeom2::LoadIdentity
IA3dGeom2::LoadMatrix
IA3dGeom2::MultMatrix
IA3dGeom2::GetMatrix

IA3dGeom2::BindMaterial

Sets the current material.

Prototype

```
HRESULT BindMaterial(
    LPA3DMATERIAL          pMaterial
);
```

Parameters

pMaterial Pointer to a material.

Return Values

S_OK
E_INVALIDARG

Description

This method sets the current material in the geometry engine. All polygons sent after **IA3dGeom2::BindMaterial** is called have the acoustic properties of that material. The parameter *pMaterial* is a pointer to a material object created with **IA3dGeom2::NewMaterial**.

Any number of materials can be bound in a frame, but a single polygon can only inherit the properties of one material.

Changing the current material is a mode change, but the performance impact is very small. However, each time a material is bound, it is stored in the frame buffer and takes up a small amount of memory. To use materials efficiently, polygons should be ordered so that those with the same material are sent to the geometry engine together, minimizing mode changes and reducing the memory footprint of the frame buffer slightly. This is only really significant when many hundreds of polygons are being rendered each frame.

See Also

IA3dGeom2::NewMaterial

IA3dGeom2::BindSource

Inserts a source into the scene hierarchy.

Prototype

HRESULT **BindSource**(
    LPA3DSOURCE2 *pSource*
);

Parameters

pSource A pointer to a source.

Return Values

S_OK
E_INVALIDARG

Description

IA3dGeom2::BindSource applies the current matrix to the source pointed to by *pSource* and transforms its position, orientation and velocity. By default a source has no transformations associated with it so its properties are in absolute world space. When a source is bound to a matrix other than the identity matrix, its properties are in the coordinate system defined by that matrix.

This method is useful when attaching a source to moving geometry. Rather than continually computing the world coordinates of the vertices for the object to which the source is attached, the object could be specified in local coordinates and moved by using **IA3dGeom2::Translate** and **IA3dGeom2::Rotate**. Binding the source to the matrix that is applied to the geometry will cause it to move with the object without the application having to calculate its world location and orientation. It also provides a simple way of positioning a source relative to the listener by using the matrix functions to position the listener and binding the source to the same matrix as the listener. When this is done, the source methods **IA3dSource2::SetPosition**, **IA3dSource2::SetOrientation** and **IA3dSource2::SetVelocity** are all relative to the listener.

When the source is in head relative transformation mode, the matrix it is bound to is relative to the listener. Using **IA3dGeom2::BindSource** in head relative mode is equivalent to loading a matrix that specifies a coordinate system with its origin at the listener, then multiplying in a source transformation matrix, and finally transforming the position, orientation and velocity set explicitly by the source methods.

If **IA3dGeom2::BindSource** is being used it should be called every frame.

See Also

IA3dGeom2::BindListener
IA3dGeom2::PushMatrix
IA3dGeom2::PopMatrix
IA3dGeom2::Translate
IA3dGeom2::Rotate
IA3dGeom2::Scale
IA3dGeom2::LoadIdentity
IA3dGeom2::LoadMatrix
IA3dGeom2::MultMatrix
IA3dGeom2::GetMatrix
IA3dSource2::Set/GetTransformMode

IA3dGeom2::Disable

Globally disables a feature in the Wavetracing engine.

Prototype

HRESULT **Disable**(
    DWORD *dwFeature*
);

Parameters

dwFeature *dwFeature* can be only *one* of the following:

A3D_OCCLUSIONS
A3D_1ST_REFLECTIONS

Return Values

S_OK
A3DERROR_FEATURE_NOT_AVAILABLE
A3DERROR_A3D_NOT_INITIALIZED

Description

IA3dGeom2::Disable disables various rendering features. When disabling a feature in this manner, the feature is globally disabled - all geometry for the frame is affected no matter at what point in the frame the method was called. To selectively disable rendering features for different parts of the scene graph, use **IA3dGeom2::SetRenderMode**.

Features must be available and enabled before they can be disabled (see **IA3d5::Init** and **IA3dGeom2::Enable** for details). *dwFeature* can be either A3D_OCCLUSIONS or A3D_1ST_REFLECTIONS. Note that *dwFeature* is *not* a bitmask, but a single value.

See Also

IA3dGeom2::Enable
IA3dGeom2::IsEnabled
IA3d5::Init

IA3dGeom2::Enable

Globally enables a feature in the Wavetracing engine.

Prototype

HRESULT **Enable**
    DWORD *dwFeature*
);

Parameters

dwFeature *dwFeature* can be only *one* of the following:

A3D_OCCULSIONS
A3D_1ST_REFLECTIONS

Return Values

S_OK
A3DERROR_FEATURE_NOT_AVAILABLE
A3DERROR_A3D_NOT_INITIALIZED

Description

IA3dGeom2::Enable enables various rendering features. When enabling a feature in this manner, the feature is globally enabled - all geometry for the frame is affected irrespective of where in the frame the method was called. To selectively enable rendering features for different parts of the scene graph, use **IA3dGeom2::SetRenderMode**.

Features must be available before they can be enabled. When initializing A3D, **IA3d5::Init** tells the audio renderer what features it would like, and the renderer returns which features it supports. Trying to enable a feature that isn't supported by the rendering platform returns an error. *dwFeature* can be either A3D_OCCLUSIONS or A3D_1ST_REFLECTIONS. Note that *dwFeature* is *not* a bitmask, but a single value.

By default, reflections and occlusions are disabled even if they were successfully requested at initialize. They must be enabled using **IA3dGeom2::Enable**.

See Also

IA3dGeom2::Disable
IA3dGeom2::IsEnabled
IA3dGeom2::Set/GetRenderMode
IA3d5::Init

IA3dGeom2::GetMatrix

Gets the current matrix on the stack.

Prototype

HRESULT **GetMatrix**(
    A3DMATRIX *pA3dMatrix*
);

Parameters

A3dMatrix The pointer to the matrix. The function fills in the data in the matrix pointed to by *A3dMatrix*.

Return Values

S_OK

Description

IA3dGeom2::GetMatrix returns the current transformation matrix. See **IA3dGeom2::PushMatrix** for an explanation of the matrix stack.

See Also

IA3dGeom2::LoadMatrix
IA3dGeom2::LoadIdentity
IA3dGeom2::Translate
IA3dGeom2::Rotate
IA3dGeom2::Scale
IA3dGeom2::MultMatrix

IA3dGeom2::IsEnabled

Returns whether or not a feature is enabled in the Wavetracing engine.

Prototype

```
HRESULT IsEnabled(
    DWORD    dwFeature
);
```

Parameters

dwFeature — *dwFeature* can be only *one* of the following:

A3D_OCCLUSIONS
A3D_1ST_REFLECTIONS

Return Values

TRUE — The feature is enabled.

FALSE — The feature is not enabled.

Description

Use **IA3dGeom2::IsEnabled** to check if a feature is enabled. The method returns TRUE (1) if the feature queried is enabled, and FALSE (0) if it is not enabled. Note that *dwFeature* is *not* a bitmask, but a value representing a single feature. See **IA3dGeom2::Enable**.

See Also

IA3dGeom2::Enable
IA3dGeom2::Disable
IA3d5::Init

IA3dGeom2::LoadIdentity

Loads an identity matrix onto the matrix stack.

Prototype

```
HRESULT LoadIdentity(
    void
);
```

Parameters

None.

Return Values

S_OK

Description

IA3dGeom2::LoadIdentity replaces the current matrix with the identity matrix. It is equivalent to calling **IA3dGeom2::LoadMatrix** with an identity matrix. In common with all matrix methods, **IA3dGeom2::LoadIdentity** affects only data sent following this call.

See Also

IA3dGeom2::LoadMatrix
IA3dGeom2::MultMatrix
IA3dGeom2::Translate
IA3dGeom2::Rotate
IA3dGeom2::Scale
IA3dGeom2::PushMatrix
IA3dGeom2::PopMatrix

IA3dGeom2::LoadMatrix

Loads an arbitrary matrix onto the matrix stack.

Prototype

HRESULT **LoadMatrix**(
    A3DMATRIX *pA3dMatrix*
);

Parameters

pA3dMatrix The pointer to the 4 × 4 matrix to be loaded. The function copies the data to which the pointer is pointing.

Return Values

S_OK

Description

IA3dGeom2::LoadMatrix replaces the current matrix with the one pointed to by *pA3dMatrix*. Use this method when an application is doing the math to compute transformation matrices — it saves the Wavetracing engine from duplicating the work by processing calls to **IA3dGeom2::Translate**, **IA3dGeom2::Rotate**, **IA3dGeom2::Scale**, etc.

When reusing matrices computed by another engine, some care has to be taken to ensure that the matrix conventions it uses are the same as in A3D, and that both are using the same coordinate convention (right or left handed). A3D matrices, as in OpenGL, are stored in column-major order as a one-dimensional array of floating-point numbers. That is, the elements of the array are mapped on the matrix as follows:

0	*4*	*8*	*12*
1	*5*	*9*	*13*
2	*6*	*10*	*14*
3	*7*	*11*	*15*

See Also

IA3dGeom2::LoadIdentity
IA3dGeom2::Translate
IA3dGeom2::Rotate
IA3dGeom2::Scale
IA3dGeom2::MultMatrix
IA3dGeom2::PushMatrix
IA3dGeom2::PopMatrix
IA3d5::Set/GetCoordinateSystem

IA3dGeom2::MultMatrix

Multiplies the current matrix by an arbitrary matrix.

Prototype

HRESULT **MultMatrix**(
    A3DMATRIX *pA3dMatrix*
);

Parameters

pA3dMatrix Pointer to a 4 × 4 matrix.

Return Values

S_OK

Description

IA3dGeom2::MultMatrix multiplies the current matrix with the one specified in *pA3dMatrix*. This replaces the current matrix, M, with M * *pA3dMatrix*. See **IA3dGeom2::LoadMatrix** for more information on matrix operations.

See Also

IA3dGeom2::LoadIdentity
IA3dGeom2::Translate
IA3dGeom2::Rotate
IA3dGeom2::Scale
IA3dGeom2::LoadMatrix
IA3dGeom2::PushMatrix
IA3dGeom2::PopMatrix

IA3dGeom2::NewList

Creates a new list of geometry data.

Prototype

```
HRESULT NewList(
    LPA3DLIST *ppList
);
```

Parameters

ppList The address of a list pointer. The function fills out the list pointer.

Return Values

A3D_OK
A3DERROR_MEMORY_ERROR
A3DERROR_INVALID_ARGUMENT

Description

A list is an object used to record a sequence of **IA3dGeom2** methods. **IA3dGeom2::NewList** is used to create a new list object and return a pointer to its interface. Once a sequence of **IA3dGeom2** methods has been stored in a list, that list can be executed many times. Since the result of the **IA3dGeom2** methods is stored rather than the commands, executing a list is dramatically faster than sending the individual methods that were used to create the list. A pointer to the transformed and packed data is simply inserted into the frame buffer.

Any number of lists can be created and executed, though only one can be recorded at a time. See the section on **IA3dList** for more details.

See Also

IA3dList::Release
IA3dList::Call
IA3dList::Begin, IA3dList::End

IA3dGeom2::NewMaterial

Creates a new material.

Prototype

HRESULT **NewMaterial(**
    LPA3DMATERIAL ***ppMaterial***
);

Parameters

ppMaterial	The address of the material pointer. The function fills out the value of the pointer.

Return Values

S_OK
E_INVALIDARG

Description

IA3dGeom2::NewMaterial creates a new acoustic material and returns a pointer to its interface in *ppMaterial*. A material has reflectance and transmittance properties that define how surfaces using the material reflect and transmit sound. By default, when a material is created, it has properties that make it a perfect reflector and occluder.

Many materials can be created though only one is active at a time. See **IA3dGeom2::BindMaterial** and the section on the **IA3dMaterial** interface for more information on how materials are applied to the geometry in a scene.

See Also

IA3dGeom2::BindMaterial
IA3dMaterial::Release
IA3dMaterial::Set/GetReflectance
IA3dMaterial::Set/GetTransmittance

IA3dGeom2::Normal

Specifies the normal for a polygon.

Prototype

```
HRESULT Normal3f(
    A3DVAL          vx, A3DVAL vy, A3DVAL vz
);

HRESULT Normal3fv(
    LPA3DVAL        pxyz
);
```

Parameters

vx, *vy*, *vx*	Three floating-point numbers specifying the normal vector to the polygon.
pxyz	A pointer to an array of 3 values which represent the vertex position.

Return Values

S_OK
E_INVALIDARG

Description

Use **IA3dGeom2::Normal** between **IA3dGeom2::Begin** and **IA3dGeom2::End** blocks to send the normals for primitives to the Wavetracing engine. The normals are transformed according to the current matrix on the stack. While this method can be called before every call to **IA3dGeom2::Vertex**, acoustic surfaces are flat shaded. The last normal to be sent is the one that will be applied to the entire polygon.

Using this method is not mandatory — if no normal is specified for a polygon, one is computed automatically.

See Also

IA3dGeom2::Vertex

IA3dGeom2::PopMatrix

Pops a matrix off the matrix stack.

Prototype

```
HRESULT PopMatrix(
    void
);
```

Parameters

None.

Return Values

S_OK
E_FAIL

Description

The Wavetracing geometry engine maintains a stack of 32 matrices for geometry transformations. The current matrix is the matrix at the top of the stack and is the one used to transform all geometry, source and listener data.

IA3dGeom2::PopMatrix pops the matrix stack up one by replacing the current matrix with the one below it in the stack. This is used to restore the current matrix to the state it was in when **IA3dGeom2::PushMatrix** was called.

An application should have an equal number of calls to **IA3dGeom2::PushMatrix** and **IA3dGeom2::PopMatrix**. If **IA3dGeom2::PopMatrix** is called when the top of the stack has already been reached the method will return an error.

See Also

IA3dGeom2::PushMatrix

IA3dGeom2::PushMatrix

Pushes a matrix onto the matrix stack.

Prototype

```
HRESULT PushMatrix(
    void
);
```

Parameters

None.

Return Values

S_OK
E_FAIL

Description

The Wavetracing geometry engine maintains a stack of 32 matrices for geometry transformations. The current matrix is the matrix at the top of the stack and is the one used to transform all geometry, source and listener data.

IA3dGeom2::PushMatrix copies the current matrix and pushes the stack down one. This means the current matrix and the one immediately below it in the stack are identical. This method is used to save the current matrix so that it can be modified by other matrix methods then later restored with a call to **IA3dGeom2::PopMatrix**.

An application should have an equal number of calls to **IA3dGeom2::PushMatrix** and **IA3dGeom2::PopMatrix**. If **IA3dGeom2::PushMatrix** is called when the stack is full the method will return an error.

See Also

IA3dGeom2::PopMatrix

IA3dGeom2::QueryInterface

Returns an interface pointer for a supported interface.

Prototype

```
HRESULT QueryInterface(
    REFIID          iid,
    LPVOID FAR      *pInterface
);
```

Parameters

iid	Interface identifier. Specify only IID_IA3dGeom2
pInterface	Address of a pointer to an interface which will be filled out by the method

Return Values

S_OK
E_NOINTERFACE

Description

All A3D interfaces inherit the **IUnknown** interface that contains a method called **Query-Interface**. This method is used to let the application know what other interfaces a particular interface supports, and to return a pointer to a requested interface if it is supported. The different A3D interfaces support different interfaces.

The **IA3dGeom2** interface doesn't support any other interfaces, so the only valid value for *iid* is IID_IA3dGeom2 which will return another geometry interface pointer and increment the reference count.

Calling any **QueryInterface** and asking for an interface that isn't supported will return the error E_NOINTERFACE. The address of the pointer passed in to the method will be left at the value it was set to by the calling method, so it may not be NULL. For this reason, it is essential to check the return value of this method.

See Also

IA3dGeom2::AddRef
IA3dGeom2::Release

IA3dGeom2::Release

Decrements the IA3dGeom2 reference count.

Prototype

```
ULONG Release(
    void
);
```

Parameters

None.

Return Values

Returns the new reference count.

Description

When going through a COM method such as **QueryInterface** or **NewSource** to get an interface pointer to a component, the reference count of the component is automatically incremented. The reference count is used to let the component know when nothing is accessing it anymore and that it can delete itself from memory.

Calling **IA3dGeom2::Release** decrements the reference count for the **IA3dGeom2** interface, and if it is 0, the object deletes itself from memory.

Note that **IA3d5**, **IA3dGeom2**, and **IA3dListener** all share the same reference count as they are simply different interfaces into the same base object. Only when all three have been released will the reference count of any one of them be 0.

All A3D 3.0 interfaces inherit the COM **IUnknown** interface that contains the methods **AddRef**, **QueryInterface**, and **Release**. *Inside COM* by Microsoft Press is an excellent resource for detailed information on COM.

See Also

IA3dGeom2::AddRef
IA3dGeom2::QueryInterface

IA3dGeom2::Rotate

Applies a rotational transformation to the current matrix.

Prototype

HRESULT **Rotate3f**(
    A3DVAL *fAngle*, A3DVAL *fx*, A3DVAL *fy*, A3DVAL *fz*
);

HRESULT **Rotate3fv**(
    A3DVAL *fAngle*, LPA3DVAL *fxyz*
);

Parameters

fAngle	Amount of angle to rotate in degrees.
fx, fy, fz	Vector about which the rotation should be performed.
fxyz	Pointer to an array of 3 values that represent the rotation axis.

Return Values

S_OK

Description

IA3dGeom2::Rotate applies a geometric transformation to the current matrix. It rotates the current coordinate system counter-clockwise by *fAngle* degrees about the vector from the origin to the point (*fx, fy, fz*). If M is the current matrix and R the matrix specified by the rotation, the current matrix is replaced with M * R. All subsequent geometry, listener and source data will be relative to this new coordinate system.

For azimuth rotations, the rotation vector is (0, 1, 0). Pitch (elevation) is a rotation about (1, 0, 0) and roll a rotation about (0, 0, 1). The matrix for each of these three basic rotations is computed slightly faster than the matrix for an arbitrary rotation axis.

See Also

IA3dGeom2::Translate
IA3dGeom2::Scale
IA3dGeom2::PopMatrix
IA3dGeom2::PushMatrix
IA3dGeom2::GetMatrix

IA3dGeom2::Scale

Applies a scale transformation to the current matrix.

Prototype

```
HRESULT Scale3f(
    A3DVAL          fx, A3DVAL fy, A3DVAL fz
);

HRESULT Scale3fv(
    LPA3DVAL        fxyz
);
```

Parameters

fx, *fy*, *fx*	Scale factor for each axis.
fxyz	A pointer to an array of 3 values which represent the x, y, and z scale factors.

Return Values

S_OK
E_INVALIDARG

Description

IA3dGeom2::Scale applies a geometric transformation to the current matrix. It scales the current coordinate system according to the values specified by (*fx*, *fy*, *fz*). If M is the current matrix and S the matrix specified by the scaling, the current matrix is replaced with M * S. All subsequent geometry, listener and source data will be relative to this new coordinate system.

See Also

IA3dGeom2::GetMatrix
IA3dGeom2::PopMatrix
IA3dGeom2::PushMatrix
IA3dGeom2::Rotate
IA3dGeom2::Translate

IA3dGeom2::Set/GetOcclusionMode

Unsupported

Prototype

```
HRESULT SetOcclusionMode(
    DWORD dwMode,
);

HRESULT GetOcclusionMode(
    LPDWORD pdwMode,
);
```

Parameters

dwMode

Return Values

A3DERROR_UNIMPLEMENTED_FUNCTION

Description

None.

See Also

None.

IA3dGeom2::Set/GetOcclusionUpdateInterval

Sets the number of frames between occlusion processing.

Prototype

HRESULT **SetReflectionUpdateInterval**(
    DWORD *dwInterval*
);

HRESULT **GetReflectionUpdateInterval**(
    LPDWORD *dwInterval*
);

Parameters

dwInterval DWORD specifying the number of frames between updates

Return Values

S_OK
E_INVALIDARG

Description

This method is used to spread occlusion processing over several frames, reducing the load imposed on the CPU by the Wavetracing engine. It does this by reusing the same occlusion factors for a source from the previous frame, bypassing all occlusion geometry processing inside **IA3d5::Flush**.

dwInterval defaults to 1, which means occlusions are computed every frame. With applications running faster than 30Hz, this can safely be set to 2 without any noticeable difference in audio quality.

Occlusion processing is spread as evenly as possible over the update interval. For an update interval of *dwInterval*, occlusions for 1/*dwInterval* sources will be computed each frame. This is to help the application maintain a consistent frame rate.

The effect on audio quality at longer update intervals is much more noticeable than with the reflection counterpart of this method. Whereas it is difficult to hear that a few reflections aren't quite in the right place, it is very easy to notice that a sound is occluded when it shouldn't be, especially if the object making the sound is visible. Because of this, some care needs to be exercised when using long occlusion update intervals. Intervals that represent update rates below 15 Hz should be avoided.

CPU usage of the Wavetracing engine is inversely proportional to the reflection and occlusion update intervals.

See Also

IA3dGeom2::Set/GetReflectionUpdateInterval
IA3d5::Flush

IA3dGeom2::SetOpeningFactor

Sets the opening factor for subfaces.

Prototype

```
HRESULT SetOpeningFactorf(
    A3DVAL          fFactor
);

HRESULT SetOpeningFactorfv(
    LPA3DVAL        pfFactor
);
```

Parameters

fFactor	Floating point number which specifies the opening factor.
pfFactor	Address of a floating point number.

Return Values

S_OK

Description

A subface is a polygon that is placed on top of a parent surface. It allows a transparency, or opening factor to be applied to the region of the parent surface it covers. This provides a simple means of putting doors or holes in large polygons without having to split the parent polygon up into several smaller polygons.

IA3dGeom2::SetOpeningFactor is used to specify the transparency of the subface. 0.0 means the area the subface covers is completely closed and the material characteristics of the parent polygon aren't modified. This is the default if this method isn't called. 1.0 means the area the subface covers is completely open or acoustically transparent. Values in between apply linearly to the parent polygon material. This method should be called before the first vertex of the subface it applies to is specified.

There are two implementations of this method. **IA3dGeom2::SetOpeningFactorf** can be used with dynamic geometry (geometry not cached in an **IA3dList**) or for openings that never change inside an **IA3dList**. This method takes the value of the opening factor and applies it directly to the subface. **IA3dGeom2::SetOpeningFactorfv** takes the address of a floating point number and performs exactly the same function as the other method, except that the address of the variable is stored with the subface rather than the explicit value. This is useful when caching geometry in lists since the opening factor can still be changed even though geometry in lists can't otherwise be modified.

See **IA3dGeom2::Begin** for more information on subfaces.

See Also

IA3dGeom2::Begin
IA3dGeom2::End
IA3dGeom2::Vertex
IA3dList

IA3dGeom2::Set/GetPolygonBloatFactor

Sets the reflection bloat factor for polygons.

Prototype

HRESULT **SetPolygonBloatFactor**(
    A3DVAL *fBloat*
);

HRESULT **GetPolygonBloatFactor**(
    LPA3DVAL *fBloat*
);

Parameters

fBloat Floating point, positive number specifying the bloat factor.

Return Values

S_OK
E_INVALIDARG

Description

This method can be used to scale individual polygons, affecting how they are considered for reflection or occlusion processing. It is a global state of the Wavetracing engine, so whatever value it is last set to is the value applied to all polygons in the frame buffer.

fBloat is the scale factor applied to all polygons and it can be any positive floating-point number. The larger the number the more likely a polygon is to reflect or occlude a source. The default of 1.0 represents no scaling.

The reflection location is not affected so the time delay is unchanged, but the amount of reflection off the polygon is affected due to the change in its area of coverage.

This method is useful when the application is selectively disabling reflections or occlusions for small polygons. To fill in for the greater number of polygons being discarded for reflections or occlusions, bloating the polygons that are sent to the Wavetracing engine helps recover much of the original scene.

IA3dGeom2::SetReflectionBloatFactor differs from the matrix operation **IA3dGeom2::Scale** in that it scales polygons as if they have a local coordinate system. The center point and plane equation are not modified.

See Also

IA3dGeom2::Scale

IA3dGeom2::Set/GetReflectionDelayScale

Sets the delay scaling factor for reflections.

Prototype

HRESULT **SetReflectionDelayScale**(
    A3DVAL *fScale*
);

HRESULT **SetReflectionDelayScale**(
    LPA3DVAL *fScale*
);

Parameters

fScale Scale factor for reflection delays.

Return Values

S_OK

E_INVALIDARG

Description

This method globally scales all reflection delays by *fScale*. It can be used to exaggerate the effect of reflections when *fScale* is greater that 1.0 (the default). *fScale* can be any positive number, but reflection delays are still clamped at the value set in **IA3d5::SetMaxReflectionDelayTime**, or the default of 0.3 seconds if that method wasn't called.

See Also

IA3dGeom2::Set/GetReflectionGainScale
IA3dSource2::Set/GetReflectionDelayScale
IA3dSource2::Set/GetReflectionGainScale

IA3dGeom2::Set/GetReflectionGainScale

Sets the gain scaling factor for reflections

Prototype

```
HRESULT SetReflectionGainScale(
    A3DVAL          fScale
);

HRESULT GetReflectionGainScale(
    LPA3DVAL        fScale
);
```

Parameters

fScale Scale factor for reflection gains.

Return Values

S_OK

E_INVALIDARG

Description

A number of factors are taken into account when the Wavetracing engine computes gain values for reflections: parent source gain setting, distance attenuation, and reflecting surface material. **IA3dGeom2::SetReflectionGainScale** globally scales all reflection gains by *fScale.* While *fScale* can be any positive number, final reflection gains are clipped at 0 dB. The default value for this setting is 1.0.

See Also

IA3dGeom2::Set/GetReflectionDelayScale
IA3dSource2::Set/GetReflectionDelayScale
IA3dSource2::Set/GetReflectionGainScale

IA3dGeom2::Set/GetReflectionMode

Unsupported.

Prototype

HRESULT **SetReflectionMode**(
    DWORD   *dwMode*,
);

HRESULT **GetReflectionMode**(
    LPDWORD *pdwMode*,
);

Parameters

dwMode

Return Values

A3DERROR_UNIMPLEMENTED_FUNCTION

Description

None.

See Also

None.

IA3dGeom2::Set/GetReflectionUpdateInterval

Sets the number of frames between reflection processing.

Prototype

HRESULT **SetReflectionUpdateInterval**(
    DWORD *dwInterval*
);

HRESULT **GetReflectionUpdateInterval**(
    LPDWORD *dwInterval*
);

Parameters

dwInterval DWORD specifying the number of frames between updates.

Return Values

S_OK
E_INVALIDARG

Description

This method is used to spread reflection processing over several frames, reducing the load imposed on the CPU by the Wavetracing engine. It does this by reusing the same reflections for a source from the previous frame, bypassing all reflection geometry processing inside **IA3d5::Flush**.

dwInterval defaults to 1, which means reflections are computed every frame. With applications running faster than 30 Hz, this can safely be set to 2 or even 4 without any noticeable difference in audio quality.

Reflection processing is spread as evenly as possible over the update interval. For an update interval of *dwInterval*, reflections for 1/*dwInterval* sources will be computed each frame. This is to help the application maintain a consistent frame rate.

CPU usage of the Wavetracing engine is inversely proportional to the reflection and occlusion update intervals.

See Also

IA3dGeom2::Set/GetOcclusionUpdateInterval
IA3d5::Flush

IA3dGeom2::Set/GetRenderMode

Sets the current render mode for polygon processing.

Prototype

```
HRESULT SetRenderMode(
    DWORD           dwMode
);

HRESULT GetRenderMode(
    LPDWORD         dwMode
);
```

Parameters

dwMode Specifies a bitmask containing the features to be enabled.

Return Values

S_OK
E_INVALIDARG
A3DERROR_FEATURE_NOT_INITIALIZED

Description

IA3dGeom2::SetRenderMode allows geometry rendering methods to be selectively enabled or disabled during a frame. It differs from **IA3dGeom2::Enable** and **IA3dGeom2::Disable** in that the modes it sets only affect geometry sent after the method has been called and isn't globally applied to the entire frame buffer.

dwMode is a bitwise OR of the features to be enabled. It accepts the symbolic constants A3D_OCCLUSIONS and A3D_1ST_REFLECTIONS. Features left out of *dwMode* will be disabled. The default state is (A3D_OCCLUSIONS | A3D_1ST_REFLECTIONS).

This method is useful for selectively disabling reflections for some polygons while still considering them for occlusions.

The absolute state of a feature in the Wavetracing engine is the AND of the state specified by this method and the global state specified by **IA3dGeom2::Enable** and **IA3dGeom2::Disable**. When each source is processed, the source render mode is ANDed with this absolute state. In short, for a feature to be enabled at any time, it must be globally enabled and it must be specified in the source and geometry rendering modes.

See Also

IA3d5::Init
IA3dGeom2::Enable
IA3dGeom2::Disable
IA3dSource2::Set/GetRenderMode

IA3dGeom2::Tag

Tags the next polygon.

Prototype

HRESULT **Tag**(
    DWORD *dwTagID*
);

Parameters

dwTagID

Return Values

S_OK

Description

This method is used to assign a unique 32 bit ID, *dwTagID*, to a polygon, allowing that polygon to be identified by the Wavetracing engine from one frame to the next. This facilitates smooth reflection blending between frames and removes the need to send polygons in the same order every frame. Calling this method when rendering reflections is mandatory - reflections can't be rendered for polygons that don't have an ID assigned to them.

IA3dGeom2::Tag should be called before the vertices defining the primitive are sent using **IA3dGeom2::Vertex**. Since multiple primitives can be sent inside a single begin/end block, **IA3dGeom2::Tag** should be called before the first vertex and then every *n* vertices, where *n* is the number of vertices in the primitive type being constructed.

It is not necessary to call this method if only occlusion processing is required.

See Also

IA3dGeom2::Begin
IA3dGeom2::End
IA3dGeom2::Vertex
IA3d5::Init

IA3dGeom2::Translate

Applies a translation to the current matrix.

Prototype

HRESULT **Translate3f**(
    A3DVAL *fx*, A3DVAL *fy*, A3DVAL *fz*
);

HRESULT **Translate3fv**(
    LPA3DVAL *fxyz*
);

Parameters

fx, *fy*, *fx*	x, y and z components of a translation vector.
fxyz	A pointer to an array of 3 values which specify a translation vector.

Return Values

S_OK

Description

IA3dGeom2::Translate applies a geometric transformation to the current matrix. It moves the origin of the current coordinate system to the point (*fx, fy, fz*). If M is the current matrix and T the matrix specified by the translation, the current matrix is replaced with M * T. All subsequent geometry, listener and source data will be relative to this new coordinate system.

See Also

IA3dGeom2::GetMatrix
IA3dGeom2::PopMatrix
IA3dGeom2::PushMatrix
IA3dGeom2::Rotate
IA3dGeom2::Scale

IA3dGeom2::Vertex

Send vertex data for a primitive to the rendering engine.

Prototype

```
HRESULT Vertex3f(
    A3DVAL          fx, A3DVAL fy, A3DVAL fz
);

HRESULT Vertex3fv(
    LPA3DVAL        fxyz
);
```

Parameters

vx, *vy*, *vx*	Position of the vertex.
pxyz	A pointer to an array of 3 values which represent the vertex position.

Return Values

S_OK

Description

Use **IA3dGeom2::Vertex** between **IA3dGeom2::Begin** and **IA3dGeom2::End** blocks to send primitive vertices to the Wavetracing engine. The vertices are transformed according to the current matrix on the stack. See **IA3dGeom2::Begin** for more information on primitive construction.

See Also

IA3dGeom2::Begin
IA3dGeom2::End
IA3dGeom2::Normal

IA3dList Interface

A *render list* is a collection of geometry engine commands, recorded and stored in an **IA3dList** for later execution. Sending the commands directly to the geometry engine results in host processing while storing the commands in a render list and executing the list later eliminates most of this processing.

For example, if you render a triangle using the geometry engine, you make five method calls: **Begin**, **Vertex**, **Vertex**, **Vertex**, and **End**. These five calls result in the following processing:

- Each vertex is transformed by the current matrix on the stack.
- A normal is computed if it isn't sent by the application.
- Resulting polygon is created and stored in the final frame buffer.

If the geometry and transformation matrix don't change, then you can eliminate this processing by storing the command sequence in a render list. Data is stored in the **IA3dList** exactly as it would be in the final frame buffer so the cost of executing a list is very small — a single pointer assignment, in fact. This still results in geometry being added to the final frame buffer, however, so the polygons in the list still impact rendering time when occlusions or reflections are enabled.

Overall, lists provide the following advantages:

- Reduced computation time
- Clearer code

These advantages greatly outweigh the single disadvantage — lists are immutable. With one exception, data in a list can never change. The exception to this rule is the opening factor applied to a subface. This is stored as a pointer so that doors in static lists can be opened and closed without regenerating the list.

Using a Render List in an Example

You create a render list with the **IA3dGeom2::NewList** method, which returns a pointer to an **IA3dList** interface. All **IA3dGeom2** methods executed inside an **IA3dList::Begin/IA3dList::End** block are executed and the results stored in the list in the order they were issued. To add the geometry stored in the list to the frame buffer, use the **IA3dList::Call** method.

A simple example that creates four polygons 1 meter apart looks like this:

```
pA3dGeom->NewList(&pFourWalls);
pFourWalls->Begin();
   pA3dGeom->LoadIdentity();
   pA3dGeom->Translate3f(20.0f, 0.0f, -20.0f);
   pA3dGeom->Rotate3f(30.0f, 0.0f, 1.0f, 0.0f);
   for (nLoop = 0; nLoop < 4; nLoop++)
   {
      pA3dGeom->Begin(A3D_QUADS);
         pA3dGeom->Tag(nLoop+1); /* never use a tag of 0 */
         pA3dGeom->Vertex3f(-2.0f, 0.0f, ((float)nLoop*4.0f)-
         2.0f);
         pA3dGeom->Vertex3f( 2.0f, 0.0f, ((float)nLoop*4.0f)-
         2.0f);
         pA3dGeom->Vertex3f( 2.0f, 2.0f, ((float)nLoop*4.0f)-
         2.0f);
         pA3dGeom->Vertex3f(-2.0f, 2.0f, ((float)nLoop*4.0f)-
         2.0f);
      pA3dGeom->End();
   }
pFourWalls->End();
```

To render the geometry later during the main **Clear**/**Flush** loop only requires this:

```
pFourWalls->Call();
```

Using a render list in this example has several advantages:

- Provides a convenient way for storing geometry in an object and giving that object a useful name, it reduces the number of function calls issued during the frame update
- Eliminates all the transformation calculations that would be necessary such as building the matrix, applying that matrix to each vertex, and computing a normal for each polygon
- Eliminates the cost of the math ((float)nLoop*4.0f)-2.0f. This is not a very expensive set of instructions, perhaps, but this is a simple example.

IA3dList::AddRef

Increments the IA3dList reference count.

Prototype

ULONG **AddRef**(
    void
);

Parameters

None.

Return Values

Returns the new reference count.

Description

When going through a COM method such as **QueryInterface** or **NewSource** to get an interface pointer to a component, the reference count of the component is automatically incremented. The reference count is used to let the component know when nothing is accessing it anymore and that it can delete itself from memory.

Whenever an interface pointer is assigned to another interface pointer, the **AddRef** method should be called to let the component know that two pointers are using the same interface. Now when the **Release** method is called, the component won't delete itself since it has been told something else is still using it. Consider the following example:

```
hr = pRoot->QueryInterface(IID_IBox, (void **)&pBox1);
if (SUCCEEDED(hr))
{
   pBox1->DrawIt();
   pBox2 = pBox1;
   pBox2->AddRef();
   pBox1->Release();
}
```

While *pBox1* is now invalid because it has been released, *pBox2* remains intact and can still be used.

All A3D 3.0 interfaces inherit the COM **IUnknown** interface that contains the methods **AddRef**, **QueryInterface**, and **Release**. *Inside COM* by Microsoft Press is an excellent resource for detailed information on COM.

See Also

IA3dList::QueryInterface
IA3dList::Release

IA3dList::Begin, IA3dList::End

Record data in a render list.

Prototype

```
HRESULT Begin(
    void
);

HRESULT End(
    void
);
```

Parameters

None.

Return Values

nNumPolygons Returns the number of polygons stored in the list.

E_FAIL

Description

An **IA3dList** object is used to store a list of **IA3dGeom2** commands. Between the calls to **IA3dList::Begin** and **IA3dList::End,** calls to **IA3dGeom2** methods are recorded and stored in the list object in the order they were issued. That list of commands can be executed later by calling **IA3dList::Call**. However, only one list can be recorded at a time, and **IA3dList::Call** can't be used to execute another list inside a begin/end block. Since it's the result of the sequence of commands that is stored rather than the commands themselves, there are significant performance advantages to using lists. These are discussed in the introduction to this section.

Lists can't be changed, so once **IA3dList::End** is called, the list will be stored as it is until it is released. The single exception to the immutability of lists is with subfaces — while the list is being recorded, if **IA3dGeom2::SetOpeningFactorfv** is used to set the transparency of subfaces, that value can be modified dynamically since the address of the opening factor variable is stored rather than its value.

Certain IA3dGeom2 methods are not recorded in the list object but instead are executed immediately. These are: IA3dGeom2::Enable, IA3dGeom2::Disable, IA3dGeom2::IsEnabled, IA3dGeom2::SetReflectionGainScale, IA3dGeom2::SetReflectionDelayScale, IA3dGeom2::SetPolygonBloatFactor, IA3dGeom2::SetReflectionUpdateInterval, IA3dGeom2::SetOcclusionUpdateInterval,

IA3dGeom2::BindListener, IA3dGeom2::BindSource and all IA3dGeom2::Get* methods.

See **IA3dGeom2::NewList** for more information on lists.

See Also

IA3dGeom2::NewList
IA3dList::Call

IA3dList::Call

Executes the sequence of commands stored in a list object.

Prototype

```
HRESULT Call(
    void
);
```

Parameters

None.

Return Values

nNumPolygons Returns the number of polygons executed by the list.

Description

IA3dList::Call sends the result of executing all the commands stored in the list to the Wavetracing engine. Geometry stored in the list object isn't modified by the current matrix but instead is rendered exactly as it was recorded.

See Also

IA3dList::Begin
IA3dList::End
IA3dGeom2::NewList

IA3dList::EnableBoundingVol

Enables bounding box culling for a list.

Prototype

HRESULT **EnableBoundingVol**(
    void
);

Parameters

None.

Return Values

S_OK

Description

IA3dList::EnableBoundingVol can be used to enable a bounding volume calculation while the list is being recorded. This volume is used when **IA3dList::Call** is issued to quickly determine if the geometry inside the list object should be considered for occlusion processing or instead trivially rejected.

To make best use of this optimization, a list should be made up of cohesive geometry. This allows the Wavetracing engine to quickly narrow down the polygons it needs to consider for occlusion testing and can easily lead to a 10x increase in polygon throughput.

If the number of polygons in the list is less than 5, this optimization is bypassed, as at that point the test itself becomes a significant percentage of testing the individual polygons.

Note that this feature is not enabled by default. This is to avoid doing the same checks an application might do if it has its own list management routines.

See Also

IA3dGeom2::NewList
IA3dList::Begin
IA3dList::End

IA3dList::QueryInterface

Returns an interface pointer for a supported interface.

Prototype

```
HRESULT QueryInterface(
    REFIID          iid,
    LPVOID FAR      *pInterface
);
```

Parameters

iid	Interface identifier. Specify only IID_IA3dList.
pInterface	Address of a pointer to an interface which will be filled out by the method

Return Values

S_OK
E_NOINTERFACE

Description

All A3D interfaces inherit the **IUnknown** interface that contains a method called **QueryInterface**. This method is used to let the application know what other interfaces a particular interface supports, and to return a pointer to a requested interface if it is supported. The different A3D interfaces support different interfaces.

The **IA3dList** interface doesn't support any other interfaces, so the only valid value for *iid* is IID_IA3dListener which will return another list interface pointer and increment the reference count.

Calling any **QueryInterface** and asking for an interface that isn't supported will return the error E_NOINTERFACE. The address of the pointer passed in to the method will be left at the value it was set to by the calling method, so it may not be NULL. For this reason, it is essential to check the return value of this method.

See Also

IA3dList::AddRef
IA3dList::Release

IA3dList::Release

Decrements the IA3dList reference count.

Prototype

```
ULONG Release(
    void
);
```

Parameters

None.

Return Values

Returns the new reference count.

Description

When going through a COM method such as **QueryInterface** or **NewSource** to get an interface pointer to a component, the reference count of the component is automatically incremented. The reference count is used to let the component know when nothing is accessing it anymore and that it can delete itself from memory.

Calling **IA3dList::Release** decrements the reference count for the **IA3dList** interface, and if it is 0, the object deletes itself from memory.

All A3D 3.0 interfaces inherit the COM **IUnknown** interface that contains the methods **AddRef**, **QueryInterface**, and **Release**. *Inside COM* by Microsoft Press is an excellent resource for detailed information on COM.

See Also

IA3dList::AddRef
IA3dList::QueryInterface

IA3dMaterial Interface

A material defines the acoustic properties of a surface — for example, a floor can be covered in tile or carpet, a wall can have wallpaper, paint, or paneling, and so on. Both light and sound behave differently depending on the material covering an object — light reflects sharply off of a shiny tile surface as opposed to a thick gray carpet just as sound bounces off of tiled floors and is absorbed by thick carpet on the floor. By specifying the properties of a material, you determine how sound will interact with polygons rendered with that material.

To specify the properties of a material, define the following:

- Reflectance

  If both the listener and the sound are on the same side of the polygon, how much sound does the listener hear? You use the **IA3dMaterial::Set/GetReflectance** method to provide this information.

- Transmittance

  If the polygon is between the listener and the sound, how much sound does the listener hear? You use the **IA3dMaterial::Set/GetOcclusion** to provide this information.

Each of these properties has two controls:

- Overall attenuation

  Similar to the gain control on sources, this determines how much the entire signal is attenuated.

- High-frequency content

  Similar to the eq control on sources, this determines how much high frequencies are attenuated.

Any number of materials can be created and applied to a scene.

Using Materials in an Example

You create a material with the **IA3dGeom2::NewMaterial** method, which returns a pointer to an **IA3dMaterial** interface:

```
IA3dMaterial *pBrick;
IA3dMaterial *pCarpet;
```

```
pIA3dGeom2->NewMaterial(&pBrick);
pIA3dGeom2->NewMaterial(&pCarpet);
```

With material objects now created, their acoustic properties can be specified:

```
pBrick->SetTransmittance(0.2f, 0.5f);
pBrick->SetRelfectance(0.9f, 0.8f);
pCarpet->SetTransmittance(0.95f, 0.6f);
pCarpet->SetReflectance(0.4f, 0.2f);
```

This is all that is required to fully define the materials and they are ready to be applied to geometry. Inside the main **IA3d5::Clear/IA3d5::Flush** block, the current material is set by issuing this instruction:

```
pIA3dGeom2->BindMaterial(pBrick);
pIA3dGeom2->Begin(A3D_QUADS);
   pIA3dGeom2->Tag(1);
   pIA3dGeom2->Vertex3fv(vert_1);
   …
```

From this point on in the frame, all polygons sent have the properties of *pBrick*. Calling **IA3dGeom2::BindMaterial** again with a different parameter will update the current material which will be applied to the polygons sent after this call to the method. It doesn't change the material properties of the polygons already sent.

If the properties of a material are modified, they don't take effect until **IA3dGeom2::BindMaterial** is used to select the material, even if it is already the current material when its properties are modified. Changing the current material does not change the properties of polygons inside a list.

IA3dMaterial::AddRef

Increments the IA3dMaterial reference count.

Prototype

ULONG **AddRef**(
    void
);

Parameters

None.

Return Values

Returns the new reference count.

Description

When going through a COM method such as **QueryInterface** or **NewSource** to get an interface pointer to a component, the reference count of the component is automatically incremented. The reference count is used to let the component know when nothing is accessing it anymore and that it can delete itself from memory.

Whenever an interface pointer is assigned to another interface pointer, the **AddRef** method should be called to let the component know that two pointers are using the same interface. Now when the **Release** method is called, the component won't delete itself since it has been told something else is still using it. Consider the following example:

```
hr = pRoot->QueryInterface(IID_IBox, (void **)&pBox1);
if (SUCCEEDED(hr))
{
   pBox1->DrawIt();
   pBox2 = pBox1;
   pBox2->AddRef();
   pBox1->Release();
}
```

While *pBox1* is now invalid because it has been released, *pBox2* remains intact and can still be used.
All A3D 3.0 interfaces inherit the COM **IUnknown** interface which contains the methods **AddRef**, **QueryInterface**, and **Release**. *Inside COM* by Microsoft Press is an excellent resource for detailed information on COM.

See Also

IA3dMaterial::QueryInterface
IA3dMaterial::Release

IA3dMaterial::Duplicate

Unsupported.

Prototype

HRESULT **Duplicate**(
    LPA3DMATERIAL **ppMaterial*
);

Parameters

ppMaterial

Return Values

A3DERROR_UNIMPLEMENTED_FUNCTION

Description

None.

See Also

None.

IA3dMaterial::GetClosestPreset

Unsupported.

Prototype

HRESULT **GetClosestPreset**(
    LPDWORD *dwPreset*
);

Parameters

dwPreset

Return Values

A3DERROR_UNSUPPORTED_FUNCTION

Description

None.

See Also

None.

IA3dMaterial::Load

Unsupported.

Prototype

HRESULT **Load**(
    LPSTR *szFileName*
);

Parameters

szFileName

Return Values

A3DERROR_UNIMPLEMENTED_FUNCTION

Description

None.

See Also

None.

IA3dMaterial::QueryInterface

Returns an interface pointer for a supported interface.

Prototype

```
HRESULT QueryInterface(
    REFIID          iid,
    LPVOID FAR      *pInterface
);
```

Parameters

iid	Interface identifier. Specify only IID_IA3dMaterial.
pInterface	Address of a pointer to an interface which will be filled out by the method

Return Values

S_OK
E_NOINTERFACE

Description

All A3D interfaces inherit the **IUnknown** interface which contains a method called **QueryInterface**. This method is used to let the application know what other interfaces a particular interface supports, and to return a pointer to a requested interface if it is supported. The different A3D interfaces support different interfaces.

The **IA3dMaterial** interface doesn't support any other interfaces, so the only valid value for *iid* is IID_IA3dMaterial which will return another material interface pointer and increment the reference count.

Calling any **QueryInterface** and asking for an interface that isn't supported will return the error E_NOINTERFACE. The address of the pointer passed in to the method will be left at the value it was set to by the calling method, so it may not be NULL. For this reason, it is essential to check the return value of this method.

See Also

IA3dListener::AddRef
IA3dListener::Release

IA3dMaterial::Release

Decrements the IA3dMaterial reference count.

Prototype

```
ULONG Release(
    void
);
```

Parameters

None.

Return Values

Returns the new reference count.

Description

When going through a COM method such as **QueryInterface** or **NewSource** to get an interface pointer to a component, the reference count of the component is automatically incremented. The reference count is used to let the component know when nothing is accessing it anymore and that it can delete itself from memory.

Calling **IA3dMaterial::Release** decrements the reference count for the **IA3dMaterial** interface, and if it is 0, the object deletes itself from memory.

All A3D 3.0 interfaces inherit the COM **IUnknown** interface which contains the methods **AddRef**, **QueryInterface**, and **Release**. *Inside COM* by Microsoft Press is an excellent resource for detailed information on COM.

See Also

IA3dMaterial::AddRef
IA3dMaterial::QueryInterface

IA3dMaterial::Save

Unsupported.

Prototype

HRESULT **Save**(
    LPSTR *szFilename*
);

Parameters

szFileName

Return Values

A3DERROR_UNIMPLEMENTED_FUNCTION

Description

None.

See Also

None.

IA3dMaterial::SelectPreset

Unsupported.

Prototype

```
HRESULT SelectPreset(
    DWORD    nMaterialEnum
);
```

Parameters

nMaterialEnum

Return Values

A3DERROR_INVALID_ENUM_MATERIAL

Description

None.

See Also

None.

IA3dMaterial::Serialize

Unsupported.

Prototype

```
HRESULT Serialize(
    LPVOID    *ppMem
    UINT      *puiMemSize
);
```

Parameters

ppMem

puiMemSize

Return Values

A3DERROR_UNIMPLEMENTED_FUNCTION

Description

None.

See Also

None.

IA3dMaterial::Set/GetNameID

Sets and gets the name ID of the material.

Prototype

```
HRESULT SetNameID(
    LPSTR      szNameBuff
);

HRESULT GetNameID(
    LPSTR      szNameBuff,
    INT        nNameBuffLen
);
```

Parameters

szNameBuff Pointer to the buffer to receive the name data.

nNameBuffLen Length of the name buffer to receive the name.

Return Values

S_OK
A3DERROR_INVALID_ARGUMENT
A3DERROR_INSUFFICIENT_BUFFERSIZE

Description

This method is used to assign a text name to a material.

See Also

None.

IA3dMaterial::Set/GetReflectance

Sets the reflectance of a material.

Prototype

```
HRESULT SetReflectance(
    A3DVAL      fGain,
    A3DVAL      fHighFreq
);

HRESULT GetReflectance(
    LPA3DVAL    fGain,
    LPA3DVAL    fHighFreq
);
```

Parameters

fGain	A floating point number between 0.0 and 1.0.
fHighFreq	A floating point number between 0.0 and 1.0.

Return Values

A3D_OK
A3DERROR_INVALID_ARGUMENTS
A3DERROR_INPUTS_OUT_OF_RANGE

Description

This method is used to specify the reflectance properties of a material. The values set by this method determine how a sound will reflect off a polygon. Broadband and high frequency characteristics can be controlled independently.

fGain specifies the overall signal attenuation, much like **IA3dSource2::SetGain**, with 1.0 meaning sound is reflected off the material unaffected and 0.0 meaning no sound is reflected. *fHighFreq* specifies the level of high frequencies reflected off the material and its effect is similar to **IA3dSource2::SetEq**. 1.0 means high frequencies are unaffected and 0.0 means they will be completely attenuated. If either parameter is 0.0, no sound will be reflected since attenuating high frequencies to that degree virtually eliminates the entire sound.

Updates to the properties of a material don't take effect until **IA3dGeom2::BindMaterial** is called, even if the material being modified is already the current material.

The following table shows some example values for the two parameters for a few materials:

Table 7. Example Material Reflectance

Materi l	***f(ain***	***fHi(ıFreq***
Carpet	.4	.2
Wood	.9	.9
Brick	.9	.8
Glass	.0	.0

See Also

IA3dGeom2::NewMaterial

IA3dMaterial::Set/GetTransmittance

Sets the transmittance of a material.

Prototype

```
HRESULT SetTransmittance(
    A3DVAL      fGain,
    A3DVAL      fHighFreq
);

HRESULT GetTransmittance(
    LPA3DVAL    fGain,
    LPA3DVAL    fHighFreq
);
```

Parameters

fAmount	A floating point number between 0.0 and 1.0.
fHighFreq	A floating point number between 0.0 and 1.0.

Return Values

A3D_OK
A3DERROR_INVALID_ARGUMENTS
A3DERROR_INPUTS_OUT_OF_RANGE

Description

This method is used to specify the transmittance properties of a material. The values set by this method determine how a sound will travel from one side of a polygon to the other. Broadband and high frequency characteristics can be controlled independently.

fGain specifies the overall signal attenuation, much like **IA3dSource2::SetGain**, with 1.0 meaning sound is transmitted through the material unaffected and 0.0 meaning no sound is transmitted. *fHighFreq* specifies the level of high frequencies transmitted through the material and its effect is similar to **IA3dSource2::SetEq**. 1.0 means high frequencies are unaffected and 0.0 means they will be attenuated completely. If either parameter is 0.0, no sound will be transmitted since attenuating high frequencies to that degree virtually eliminates the entire sound.

Updates to the properties of a material don't take effect until **IA3dGeom2::BindMaterial** is called, even if the material being modified is already the current material.

The following table shows some example values for the two parameters for a few materials:

Table 8. Example Material Transmittance

Materi l	***f(ain***	***fHi(ıFreq***
Carpet	(95	(60
Wood	(50	(50
Brick	(20	(50
Water	1 00	(30

See Also

IA3dGeom2::NewMaterial

IA3dMaterial::UnSerialize

Unsupported.

Prototype

HRESULT **UnSerialize**(
    LPVOID *pMem*
    UINT *uiMemSize*
);

Parameters

pMem

uiMemSize

Return Values

A3DERROR_UNIMPLEMENTED_FUNCTION

Description

None.

See Also

None.

Index

N

O

P

Q

R

S