Creative Labs, Inc.
1901 McCarthy Blvd.
Milpitas, CA 95035

Phone (408) 428 6600
Fax (408) 428 6611
www.creative.com

EAX® 4.0

Programmer's Guide

EAX® 4.0

Programmer's Guide

Revision History

1.0 21 Nov 2003 First Release

Trademarks and Service marks

Creative End-User Software License Agreement for Software Development Kit Version 2, March 2002

PLEASE READ THIS DOCUMENT CAREFULLY. YOU MUST AGREE TO THE TERMS OF THIS AGREEMENT BEFORE USING OR DOWNLOADING THE SOFTWARE AND/OR MANUAL FROM THE INTERNET. BY USING OR DOWNLOADING THE SOFTWARE AND/OR MANUAL, YOU AGREE TO BE BOUND BY THE TERMS OF THIS AGREEMENT. THIS AGREEMENT SHOULD BE PRINTED AND RETAINED FOR REFERENCE.

This is a legal agreement between you ("Licensee") and Creative Technology Ltd. and its subsidiaries ("Creative"). This Agreement states the terms and conditions upon which Creative offers to license the software and/or manual provided or downloaded from this website together with all related documentation and accompanying items including, but not limited to, the executable programs, drivers, libraries and data files associated with such programs (collectively, the "Software").

LICENSE

1. Grant of License
The Software is licensed, not sold, to you for use only under the terms of this Agreement. This License Agreement is your proof of license to exercise the rights granted herein and must be retained by you. As between you and Creative (and, to the extent applicable, its licensors). Creative retains all title to and ownership of the Software and reserves all rights not expressly granted to you. The license under this Section 1 is conditioned upon your compliance with all of your obligations under this Agreement. Creative grants to you the right to use all or a portion of this Software provided that:

a. the Software is not distributed for profit;
b. the Software may NOT be modified;
c. all copyright notices are maintained on the Software;
d. the licensee/end-user agrees to be bound by the terms of this Agreement;
e. Creative's BBS/FTP/website are the only on-line sites where Licensee may download electronic files containing the Software; and
f. Licensee shall use the Software solely for the purpose of developing Licensee applications compatible with Creative's products, unless otherwise agreed to by further written agreement from Creative.

2. For Use on a Single Computer
The Software may be used only on a single computer by a single user at any time. You may transfer the machine-readable portion of the Software from one computer to another computer, provided that:

a. the Software (including any portion or copy thereof) is erased from the first computer, and
b. there is no possibility that the Software will be used on more than one computer at a time.

3. Stand-Alone Basis
You may use the Software only on a stand-alone basis, such that the Software and the functions it provides are accessible only to persons who are physically present at the location of the computer on which the Software is loaded. You may not allow the Software or its functions to be accessed remotely, or transmit all or any portion of the Software through any network or communication line.

4. Copyright
The Software is owned by Creative and/or its licensors, and is protected by United States copyright laws and international treaty provisions. You may not remove the copyright notice from any copy of the Software or any copy of the written materials, if any, accompanying the Software.

5. One Archival Copy
You may make one (1) archival copy of the machine-readable portion of the Software for backup purposes only in support of your use of the Software on a single computer, provided that you reproduce on the copy all copyright and other proprietary rights notices included in the originals of the Software.

6. Merger or Integration
You may merge only the Software code examples provided into or integrate them with, any other program. Except as set forth in this Section 6, you may not merge or integrate any of the software with any other program, except to the extent expressly permitted by the laws of the jurisdiction where you are located. Any portion of the Software code examples merged into or integrated with another program, if any, will continue to be subject to the terms and conditions of this Agreement, and you must reproduce on the merged or integrated portion all copyright and other proprietary rights notices included on the originals of the Software.

7. Not for Use on a Network
The Software provided under this Agreement may not be used on a file server or other network device and may not be copied onto multiple systems.

8. Transfer of License
You may not transfer your license of the Software to a third party.

9. Limitations on Using and Copying the Software
Except to the extent expressly permitted by this Agreement or by any other developer agreement agreed to in writing by Creative, you may not use nor copy the Software for any purpose other than software development. Nor may you sub-license any of your rights under this Agreement. You may use the Software for your personal use only, and absent a written agreement with Creative to the contrary, not for public performance or for the creation of publicly displayed videotapes.

10. Decompiling, Disassembling, or Reverse Engineering
You acknowledge that the Software contains trade secrets and other proprietary information of Creative and its licensors. Except to the extent expressly permitted by this Agreement or by the laws of the jurisdiction where you are located, you may not decompile, disassemble or otherwise reverse engineer the Software, or engage in any other activities to obtain underlying information that is not visible to the user in connection with normal use of the Software.

In particular, you agree not to transmit the Software or display the Software's object code for any purpose on any computer screen or to make any hardcopy memory dumps for any purpose of the Software's object code. If you believe you require information related to the interoperability of the Software with other programs, you shall not decompile or disassemble the Software to obtain such information, and you agree to request such information from Creative at the address listed below. Upon receiving such a request, Creative shall determine whether you require such information for a legitimate purpose and, if so, Creative will provide such information to you within a reasonable time and on reasonable conditions.

In any event, you will notify Creative of any information derived from reverse engineering or such other activities, and the results thereof will constitute the confidential information of Creative that may be used only in connection with the Software.

TERMINATION

The license granted to you is effective until terminated. You may terminate it at any time by destroying the Software (including any portions or copies thereof) currently in your possession or control. The license will also terminate automatically without any notice from Creative if you fail to comply with any term or condition of this Agreement. You agree upon any such termination to destroy the Software (including any portions or copies thereof). Upon termination, Creative may also enforce any and all rights provided by law. The provisions of this Agreement that protect the proprietary rights of Creative will continue in force after termination.

NO WARRANTY

ANY USE BY YOU OF THE SOFTWARE IS AT YOUR OWN RISK. THE SOFTWARE IS PROVIDED FOR USE ONLY WITH CREATIVE'S HARDWARE AND RELATED SOFTWARE. THE SOFTWARE IS PROVIDED FOR USE "AS IS" WITHOUT WARRANTY OF ANY KIND TO THE MAXIMUM EXTENT PERMITTED BY LAW, CREATIVE DISCLAIMS ALL WARRANTIES OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY, OR-FITNESS FOR A PARTICULAR PURPOSE. TITLE OR NONINFRINGEMF.NT CREATIVE IS NOT OBLIGATED TO PROVIDE ANY UPDATES OR UPGRADES TO THE SOFTWARE.

No distributor, dealer or any other entity or person is authorized to expand or alter this warranty or any other provisions of this Agreement. Creative does not warrant that the functions contained in the Software will meet your requirements or that the operation of the Software will be uninterrupted, error-free, or free from malicious code. For purposes of this paragraph, “malicious code” means any program code designed to contaminate other computer programs or computer data, consume computer resources, modify, destroy, record, or transmit data, or in some other fashion usurp the normal operation of the computer, computer system, or computer network, including viruses, Trojan horses, droppers, worms, logic bombs, and the like.

Further, Creative shall not be liable for the accuracy of any information provided by Creative or third-party technical support personnel, or any damages caused, either directly or indirectly, by acts taken or omissions made by you as a result of such technical support.

Any representation, other than the warranties set forth in this Agreement, will not bind Creative. You assume full responsibility for the selection of the Software to achieve your intended results, and for the downloading, use and results obtained from the Software. You also assume the entire risk as it applies to the quality and performance of the Software. Should the Software prove defective, you (and not Creative, or its distributors or dealers) assume the entire liability of any and all necessary servicing, repair or correction.

This warranty gives you specific legal rights, and you may also have other rights, which vary from country/state to country/state. Some countries/states do not allow the exclusion of implied warranties, so the above exclusion may not apply to you. Creative disclaims all warranties of any kind if the Software was customized, repackaged, or altered in any way by any third party other than Creative.

IN NO EVENT WILL CREATIVE'S LIABILITY TO YOU OR ANY OTHER PERSON EVER EXCEED THE AMOUNT PAID BY YOU TO USE THE SOFTWARE, REGARDLESS OF THE FORM OF THE CLAIM.

NO LIABILITY FOR DAMAGES, INCLUDING WITHOUT LIMITATION CONSEQUENTIAL DAMAGES

In no event shall Creative or its Licensor's be liable for any damages whatsoever (including, without limitation, incidental, direct, indirect, special or consequential damages, damages for loss of business profits, business interruption, loss of business information, or other pecuniary loss) arising out of the use or inability to use this Software, even if Creative or its Licensor's have been advised of the possibility of such damages. Because some states/ countries do not allow the exclusion or limitation of liability for consequential or incidental damages, the above limitation may not apply to you.

INDEMNIFICATION BY YOU
If you distribute the Software in violation of this Agreement, you hereby indemnify, hold harmless and defend Creative from and against any and all claims or lawsuits, including attorney's fees and costs that arise, result from, or are connected with the use or distribution of the Software in violation of this Agreement.

U.S. GOVERNMENT RESTRICTED RIGHTS
All Software and related documentation are provided with restricted rights. Use, duplication or disclosure by the U.S. Government is subject to restrictions as set forth in subdivision (b)(3)(ii) of the Rights in Technical Data and Computer Software Clause at 252.227-7013. If you are sub-licensing or using the Software outside of the United States, you will comply with the applicable local laws of your country, U.S. export control law, and the English version of this Agreement.

CONTRACTOR/MANUFACTURER
The Contractor/Manufacturer for the Software is:

Creative Technology Ltd.
31 International Business Park
Creative Resource
Singapore 609921

Safety & Regulatory Information
The following sections contain notices for various countries:

CAUTION: This device is intended to be connected by the user to a CSA/TUV/UL certified/listed IBM AT or compatible personal computers in the manufacturer's defined operator access area. Check the equipment operating/installation manual and/or with the equipment manufacturer to verify/confirm if your equipment is suitable for devices to be connected to it.

ATTENTION: Ce périphérique est destiné à être connecté par l'utilisateur à un ordinateur IBM AT certifié ou listé CSA/TUV/UL ou compatible, à l'intérieur de la zone d'accès définie par le fabricant. Consulter le mode d'emploi/guide d'installation et/ou le fabricant de l'appareil pour vérifier ou confirmer qu'il est possible de connecter d'autres périphériques à votre système.

GENERAL
This Agreement is binding on you as well as your employees, employers, contractors and agents, and on any successors and assignees. Neither the Software nor any information derived therefrom may be exported except in accordance with the laws of the U.S. or other applicable provisions. This Agreement is governed by the laws of the State of California (except to the extent federal law governs copyrights and federally registered trademarks). This Agreement is the entire agreement between us and you agree that Creative will not have any liability for any untrue statement or representation made by it, its agents or anyone else (whether innocently or negligently) upon which you relied upon entering this Agreement, unless such untrue statement or representation was made fraudulently. This Agreement supersedes any other understandings or agreements, including, but not limited to, advertising, with respect to the Software.

If any provision of this Agreement is deemed invalid or unenforceable by any country or government agency having jurisdiction, that particular provision will be deemed modified to the extent necessary to make the provision valid and enforceable, and the remaining provisions will remain in full force and effect.

For questions concerning this Agreement, please contact Creative at the address stated above. For questions on product or technical matters, contact the Creative technical support center nearest you.

SPECIAL PROVISIONS APPLICABLE TO THE EUROPEAN UNION
If you downloaded the Software in the European Union (EU), the following provisions also apply to you. If there is any inconsistency between the terms of the Software License Agreement set out above and the following provisions, the following provisions shall take precedence.

Decompilation
You agree not for any purpose to transmit the Software or display the Software's object code on any computer screen or to make any hard copy memory dumps of the Software's object code. If you believe you require information related to the interoperability of the Software with other programs, you shall not decompile or disassemble the Software to obtain such information, and you agree to request such information from Creative at the address listed above. Upon receiving such a request, Creative shall determine whether you require such information for a legitimate purpose and, if so. Creative will provide such information to you within a reasonable time and on reasonable conditions.

Limited Warranty
EXCEPT AS STATED ABOVE IN THIS AGREEMENT, AND AS PROVIDED BELOW UNDER THE HEADING "STATUTORY RIGHTS," THE SOFTWARE IS PROVIDED AS-IS WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY, QUALITY AND FITNESS FOR A PARTICULAR PURPOSE.

Limitation of Remedy and Damages
THE LIMITATIONS OF REMEDIES AND DAMAGES IN THE SOFTWARE LICENSE AGREEMENT SHALL NOT APPLY TO PERSONAL INJURY (INCLUDING DEATH) TO ANY PERSON CAUSED BY CREATIVE'S NEGLIGENCE AND ARE SUBJECT TO THE PROVISION SET OUT BELOW UNDER THE HEADING "STATUTORY RIGHTS."

Irish Statutory rights
Irish law provides that certain conditions and warranties may be implied in contracts for the sale of goods and in contracts for the supply of services. Such conditions and warranties are hereby excluded, to the extent such exclusion, in the context of this transaction, is lawful under Irish law. Conversely, such conditions and warranties, insofar as they may not be lawfully excluded, shall apply. Accordingly, nothing in this Agreement shall prejudice any rights that you may enjoy by virtue of Sections 12, 13, 14 or 15 of the Irish Sale of Goods Act 1893 (as amended).

General
This Agreement is governed by the laws of the Republic of Ireland. The local language version of this agreement shall apply to Software downloaded in the EU. This Agreement is the entire agreement between us and you agree that Creative will not have any liability for any untrue statement or representation made by it, its agents or anyone else (whether innocently or negligently) upon which you relied upon entering this Agreement, unless such untrue statement or representation was made fraudulently.

Table of Contents

EAX 4.0

Programmer's Guide

This document describes how to develop software utilizing version 4.0 of Creative®'s EAX™ technology. EAX 4.0 can be used with OpenAL or with the Microsoft® DirectX™ game and multimedia programming environment. EAX allows applications to apply digital effects processing to 3D sounds. These effects include both 'environmental' effects such as reverberation, reflections, occlusion, and obstruction, and some studio-style effects like chorus, flanger, echo, EQ, and distortion.

This document provides programming information for EAX. For a higher-level look at environmental audio, please consult the EAX 4.0 Introduction Guide. If you are interested in acoustic design, please consult the EAX 4.0 Sound Designer's Guide.

What's new in EAX 4.0?

Version 3.0 of EAX provided solely environmental effects, in the form of a single reverberator capable of rendering environmental reverb, and per-source filtering. EAX 4.0 does not offer any significant changes to the EAX 3.0 acoustic model – to the way EAX actually simulates acoustics. However, there are two very important innovations introduced by EAX 4.0:-

- *Multiple effect slots* – There are now multiple effect slots, each of which can host a hardware-accelerated environmental reverb or studio-style effect.

- *New studio-style effect types* – There are eleven new studio-style effect types, providing customizable post-production processing on sound sources.

Having multiple effect slots available, which can hold a variety of effect types, amounts to a large change in the capabilities of EAX.

Multiple effect slots

EAX 4.0 on Creative's SoundBlaster Audigy series of hardware supports four effect slots. What's more, EAX 4.0's extendable interface can easily be updated when hardware becomes available with support for more effect processing. This in turn means that it will also be simple to update EAX 4.0 applications to support extra effect slots, new effect types and more effect sends when new hardware and drivers make them available.

With EAX 4.0, multiple environmental reverb effects can be used together. This flexibility allows the developer to create a simulation where several acoustic spaces are realistically modeled. The EAX 4.0 interface allows each sound source to feed multiple effects, too. In the current SDK release there is a practical limit; a source can only feed up to two effects. Again, the extendable nature of the programming interface introduced by EAX 4.0 means that when hardware that is even more powerful is introduced, this number could be raised.

To ensure that concurrently running EAX 4.0 applications cannot interfere with one another's effect slots, causing unpredictable results, an application can designate an effect slot as locked. A program cannot load a new effect type into a slot, which was previously locked by another application.

Studio-style effect types

The sophisticated environmental reverberation and filtering effects defined in EAX 3.0 have been carried forward largely unchanged to EAX 4.0. The provision of several effect slots opens up new possibilities for developers. However, sound designers might find it desirable to do more than simply render multiple environmental reverberation effects at once.

Studio effects are often employed by sound designers during production to adjust pre-recorded sound samples. For example, ring modulation can be used to make a cleanly recorded speech sample sound like an alien voice. With EAX 4.0, these effects can be rendered and controlled in real-time, with hardware acceleration meaning no CPU resources are used. In a driving simulator, the engine sound can now have distortion applied, with the effect's strength and tone being determined by the engine's speed.

Multi-Environment Scenario

To illustrate the advantages of EAX 4.0 multiple environments over EAX 3.0, consider the following two examples. In Figure 1, a typical EAX 3.0 scenario is shown. There are three sound sources all of which are interacting with the listener's environment. However, EAX 3.0 only renders a single environment, so it is only actually possible to hear a reverb simulating the acoustic properties of the listener's own surroundings.

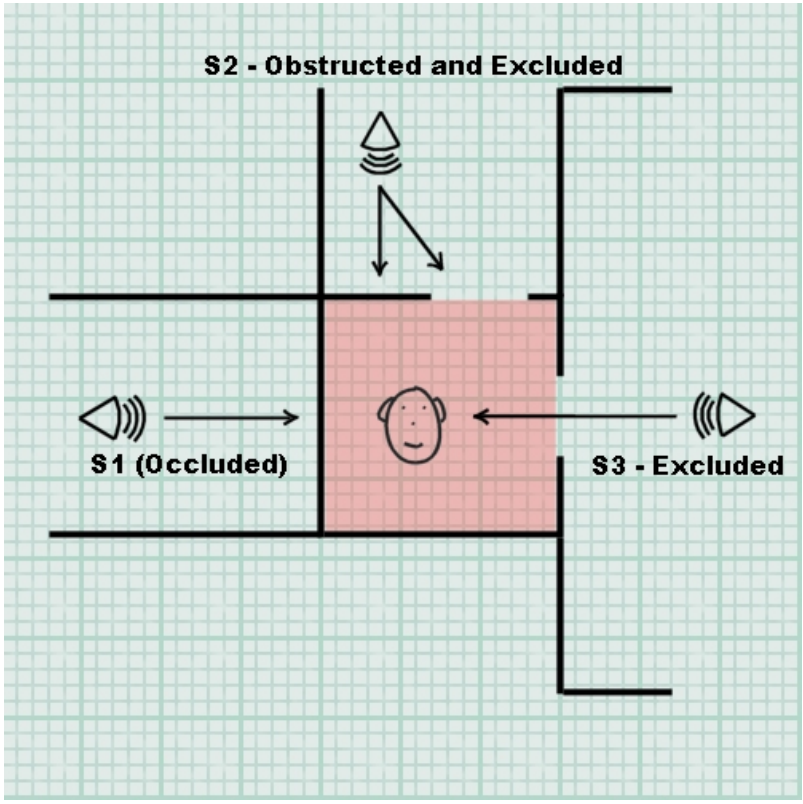

Figure 1 - EAX single environment scenario

In Figure 2, the same scenario is shown for EAX 4.0. With the ability to render multiple environments, it is possible to take into account the surroundings of the closest sources. Each source can feed its own environment as well as the listener environment. As before, the feed to the Listener's environment may need to be occluded, obstructed, or excluded based on the room geometry. In addition to setting the appropriate reverb parameters for the listener's environment, the reverb parameters for the closest rooms will need to be applied. For the non-listener environments, the reverb effect should be panned and focused to the location of the connecting wall or doorway to that room.

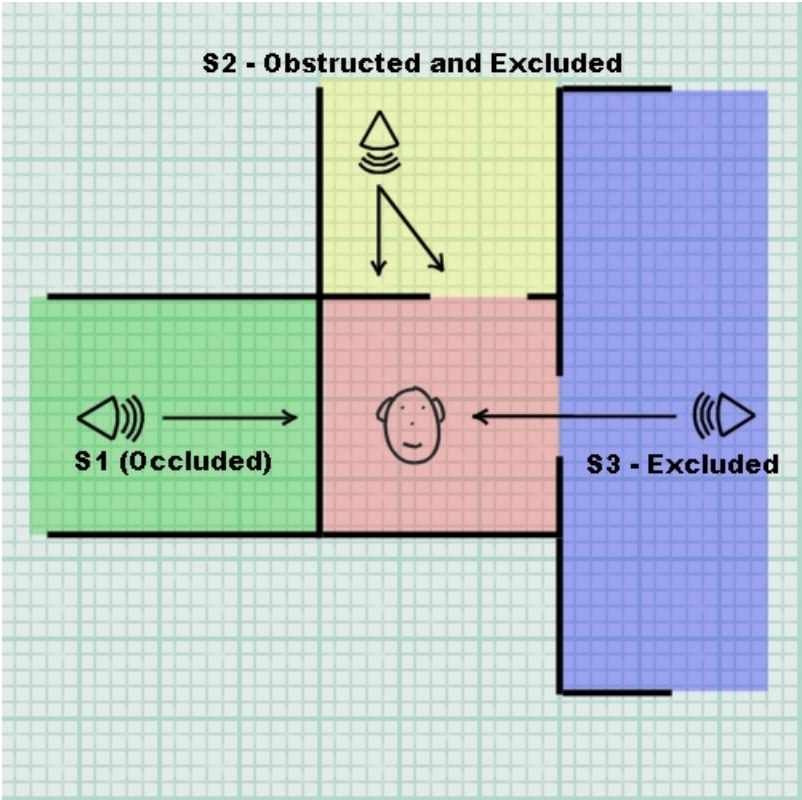

Figure 2 - EAX multiple environment scenario

How to use this guide

The EAX 4.0 programmers guide is split into six sections.

The first five sections are intended to introduce the programmer to the concepts behind the EAX interface, and instruct them on the best coding practices for implementing a powerful and robust EAX 4.0 application.

The final section is the API reference, comprising full details of every EAX 4.0 property.

What's new in EAX 4.0?

This section introduces the new features in EAX 4.0.

Introducing the EAX 4.0 Programming Interface

This section describes how the EAX 4.0 programming interface is structured, and the relationship between the interface's different objects and properties.

Programming the EAX 4.0 interface

This section demonstrates how to manipulate the EAX interface objects to perform the basic tasks necessary to control EAX 4.0 effects. The C code examples in this section are excerpts from an example audio library – the full library is distributed with the EAX 4.0 SDK.

Environmental Audio Programming Techniques

This section provides commentary on how EAX 4.0 can be integrated into an application's 3D audio engine to implement acoustic simulation with one or more environmental reverbs.

Performance and Optimization

This section details current best practices for optimizing 3D audio and EAX performance in an application.

Tables of EAX Properties

Details the properties of each EAX 4.0 object, including the minimum, maximum, and default values of each parameter. This section also includes information about the parameters of each of the effect types.

Introducing the EAX 4.0 Programming Interface

EAX 4.0 programming involves controlling the properties of three different types of object. There is one **Context object**, which is used to set global parameters including listener-specific settings. There are four **FX Slot objects**, one for each audio effect that can be rendered simultaneously by the audio hardware; these are used to manage effects. Each 3D sound source that can be rendered in hardware has a **Source object** associated with it, allowing the programmer to control how the source feeds each FX Slot, and the level of EAX source effects such as occlusion, obstruction, and filtering.

In order to use EAX 4.0 effectively, it is important to understand the relationships between the EAX 4.0 objects. By setting up the properties and relationships between all these objects, an application can determine the overall EAX 4.0 effect.

Context Object

The context object exposes all the global and listener-related properties in EAX 4.0. The following properties are contained in the Context object: -

- **Primary FX Slot ID** - Used to indicate which, if any, of the FX Slots is rendering the listener's environment. For applications without a 3D listener or applications that do not wish to use environmental audio features, this property can be safely ignored.

  For applications with a 3D listener, this parameter should be updated as the listener moves from one environment to another. This parameter serves two purposes. The first purpose is to support the legacy Occlusion and Exclusion properties in the Source object, which will apply filtering on the send to *this* FX Slot (as well as the direct / dry path of the sound source in the case of Occlusion). In EAX 3.0 Occlusion would filter the direct path of a sound, and both Occlusion and Exclusion would filter the feed into the single environment. In EAX 4.0 with potentially multiple environments, it is not known which environment feed should be filtered, unless the application sets this parameter. By adjusting this parameter, an existing application with support for Occlusion and Exclusion can continue to use their existing code. (Note: there are new EAX 4.0 Source Occlusion Send parameters that can be used to adjust Occlusion amounts to any of the FX Slots, regardless of listener location). The second purpose for this parameter is to indicate to EAX Unified (EAX *Backwards Compatibility* technology), which, if any, of the FX Slots should be translated.

- **Distance Factor** - This property is equivalent to the Direct Sound distance factor, and will combine multiplicatively with it. This property is intended for use with OpenAL applications, because OpenAL does not have this property. See the section *Distance Units* later in this document.

- **Air Absorption HF** – Controls the distance-dependant attenuation caused by the propagating medium on the direct-path sound at high frequencies. If the listener is located in a reverb environment (i.e. the FX Slot indicated by the Primary FX Slot ID property contains an EAX Reverb effect and has the Environment flag set to TRUE) then this parameter is ignored (because the environment's value is used instead, for both the direct-path sound and the reverb send path). This property is provided to allow Air Absorption effects when the listener is not located in a reverb environment.

- **HF Reference** – Controls the frequency at which the high-frequency effects created by EAX properties are measured. If the listener is in a reverb environment (as described above), this property is ignored and the value of HF Reference is taken from the environmental reverb contained in the Primary FX Slot.

FX Slot Object

The four FX Slots objects are used to manage audio effects. The FX Slot object is used to load audio effects, lock or unlock the type of effect loaded, control the effect output volume, and indicate whether the effect should be treated as 'environmental'. The FX Slot object also exposes the parameters of the loaded effect.

An FX Slot can load any one of the twelve effects that EAX 4.0 currently supports: -

- Reverb (compatible with EAX 3.0)
- AGC Compressor
- Autowah
- Chorus
- Distortion
- Echo
- Equalizer
- Flanger
- Frequency Shifter
- Vocal Morpher
- Pitch Shifter
- Ring Modulator

Note that for legacy reasons, Slot 0 will always contain the Reverb effect and Slot 1 will always contain the Chorus effect. Slots 2 and 3 can be used to store any of the audio effect types mentioned above.

The following properties are members of the FX Slot object: --

- **LoadEffect** – Each of the twelve currently defined effects types are identified by a unique GUID. Attempting to load an unidentified effect GUID will result in an error condition (see *Error Checking* later in the document). Attempting to load an effect into an FX Slot that has been locked by another application will also result in an error condition.

- **Volume** – Controls the output volume of the effect contained in the FX Slot

- **Lock** – Can be used to lock or unlock the type of effect loaded into this FX Slot. EAX 4.0 applications should be prepared for the situation where some of the FX Slots have been locked by other applications. Note that it is not possible to lock the values of the parameters of the effect. For more information, see the section on *Creating a co-operative EAX application*.

- **Flags** – Currently the only flag defined is the Environment flag, used to indicate if the FX Slot contains an environmental effect – see the section *Environmental effects* below.

- **Parameter** – Individual effect parameters can be modified through the FX Slot object. All audio effects define their parameter enumeration values in the range 0 thru 63 to avoid conflicting with the FX Slot properties defined above.

For detailed descriptions of the FX Slot properties, see the *FX Slot Properties* section in the API Reference. For detailed descriptions of the parameters available for each effect type, see the appropriate section in the *Tables of EAX Properties*.

Environmental effects

In EAX 4.0 it is necessary to indicate if an effect loaded into one of the four FX Slots is 'environmental' or not. This is handled by the Environment flag property of the FX Slot object.

An Environment effect is typically an effect that is intended to simulate a space surrounding the 3D listener or sources, like a reverb, or perhaps an echo. These effects should be affected by the physical positions of the sources and the listener. Most of the other effect types in EAX 4.0 would be considered non-environmental, as they are not related to physical positions of the sources and listener but are typically used to modify the timbre of the source.

The EAX Engine performs different calculations when computing the send levels to environmental versus non-environmental effect slots. In addition, some of the EAX Source properties are disabled for non-environmental effect slots.

The EAX Engine disables the following calculations when computing sends levels from sources to non-environment effect slots: -

- Reverb Statistical Model (if the effect is a reverb)
- Room Roll-Off Factors

The EAX Source Occlusion Sends, Exclusion Sends, Room, and Room HF parameters are also ignored when computing effect send levels to non-environmental effects.

However, any attenuation or filtering applied by the EAX engine on the direct-path sound of a source (due to distance, air absorption, source orientation or obstruction) is also applied on the effect paths into non-environmental effect slots so that, for these effect slots, the wet/dry mix is actually preserved irrespective of source and listener positions.

Source Object

Each 3D sound source has an associated Source Object, which can be used to control source-specific EAX properties such as Occlusion, Obstruction, and Exclusion. The Source object also includes properties to directly control how audio from the source feeds into the FX Slots. The new Source properties are: -

- **Send Parameters** – This property is used to set the Send and Send HF levels from the Source to a specific FX Slot. An array of structures, one for each FX Slot, can be set in one call to completely configure a Source's send levels to all the FX Slots.

- **Occlusion Send Parameters** – This property is used to set the amount of Occlusion from the source into each FX Slot. As with the Send Parameters property, an array of structures can be set in one call.

- **Exclusion Send Parameters** – Like the Occlusion Send Parameters property, this property is used to set the Exclusion amount into each FX Slot.

- **All Send Parameters** – This property can be used to completely describe the way a Source feeds all of the FX Slots. All of the above properties can be set at once using this call.

- **Active FX Slot ID** – The current implementation of EAX 4.0 allows each Source to feed up to two FX Slots simultaneously. The FX Slot sends that will be enabled are determined using this property. Send levels from the source to all FX Slots are stored by the EAX engine, but the values will only become effective when the referenced Slot is flagged as active.

  The ActiveFXSlotID property expects an array of GUIDs identifying the FX Slots that will be fed from this Source. A specially defined GUID, called EAX_PrimaryFXSlotID can be used in place of an FX Slot GUID, to indicate that the source should always feed the Listener's environment (as defined in the Context Object's PrimaryFXSlotID property). This is most useful for applications wishing to support multiple environments where it is expected that all Sources will feed their own environment, and the listener's environment. Without this special GUID, each time the Listener changed environment it would be necessary to update each Source's ActiveFXSlotID to include the GUID of the FX Slot now rendering the Listener's environment.

For detailed descriptions of all the Source object properties, see the *Source Properties* section of the API Reference.

Programming the EAX 4.0 interface

How EAX relates to 3D Audio

As discussed previously, EAX is available as an extension to both Direct Sound and OpenAL. In both APIs, EAX effects will only apply to hardware accelerated, mono, 3D sources / buffers. EAX effects are not applied to software-based buffers, or to stereo (or multi-channel) buffers.

EAX effects, especially environmental reverb, use parameter values set in the 3D Audio API. For example, the EAX engine uses the distance between source and listener to calculate the reverb send level from each sound source. Some EAX parameters will affect all audio generated by the Audio API, for example, the Context's Distance Factor parameter.

EAX and the 3D Audio API are therefore closely tied together, so it is important to understand how to correctly configure the API to best take advantage of EAX.

Voice Management

Before discussing EAX programming in more detail it is important to talk about the subject of Voice Management. It is common for an audio intensive application, such as a game, to use hundreds of audio samples. It is also possible that the application will need to play more samples simultaneously than the underlying hardware can render. These two problems can be solved with a voice management or voice allocation system.

Open AL Voice Management

Open AL was deliberately designed to separate audio samples ('Buffers') from playback voices ('Sources'). An application is free to create as many Buffers as memory allows, but the number of Sources that can be created is limited to the number of voices that the underlying Device can actually play simultaneously.

This design decision solves the first voice allocation problem by separating audio samples from hardware voices. It also allows the application developer to manage which Buffers should be played on the limited number of Sources. Whenever a Buffer needs to be played, the application can find an available Source, attach the Buffer, apply the appropriate 3D and EAX settings, and start the Source playing. If no Source is available, because they are all playing other Buffers, then the application can decide to prematurely stop a Source to make way for the new Buffer. There are various schemes that can be used for the algorithm that decides which Source to 'bump', some common schemes are; by distance, by priority, or by time left to play.

Direct Sound Voice Management

The original Direct Sound did not separate audio samples from hardware playback voices. By default, whenever a Direct Sound buffer was created, it was assigned a hardware voice. If no hardware voices were available, then the buffer was assigned a software voice. If a soundcard supported 16 3D voices then the first 16 Direct Sound buffers created with the CTRL_3D flag would be allocated hardware voices. Any buffers created after this would automatically be allocated software voices. At this point, the resources are fixed, i.e. whenever buffer 17 is played, it will always play in software regardless of whether any of the 16 hardware buffers are playing.

There are several problems associated with mixing hardware and software buffers. The most obvious is that the software voices will only be output on the front (stereo) speakers, whereas hardware voices can pan around all the speakers attached to the soundcard. Additionally, EAX effects will only apply to the hardware buffers, leaving the software buffers completely unaffected by reverb or by occlusion or obstruction.

There have been various additions to Direct Sound over the years to help solve the voice management problem, but until Direct X 9, no solution was all encompassing. Rather than detail the various additions to Direct Sound, this document will focus on the solution available in Direct X 9.

When a Direct Sound Buffer is created, it is possible to defer the allocation of the resources it will use to play by specifying the DSBCAPS_LOCDEFER flag. By deferring the resources for a buffer, it is possible to create as many buffers as the application desires.

When the application wishes to play a buffer it can specify what type of voice (hardware or software) should be used to play it. This can be done using the **Play** call, or using the **AcquireResources** call. For the purposes of EAX, it is better to use the **AcquireResources** call so that EAX Source settings can be applied before the buffer is played.

As it is only possible to play one instance of every Direct Sound Buffer, it is necessary either to create multiple copies of each Direct Sound Buffer, which is obviously extremely wasteful on memory, or to use the DuplicateSoundBuffer method whenever a buffer needs to be played.

A good solution therefore is to create a Direct Sound Buffer, using the DSBCAPS_LOCDEFER flag, for every audio sample required by the application. When the application needs to play a buffer, the Direct Sound Buffer can be duplicated, a hardware voice can be acquired (using the AcquireResources method), the 3D and EAX settings applied, and the newly duplicated buffer played.

To ease management over the hardware voices, it can be beneficial to determine at initialization time how many 3D buffers the hardware can render simultaneously. This should be done by physically creating as many hardware 3D buffers as possible, rather than looking at the DSCAPS structure. The audio engine can then expose that number of sources, and allow the calling application to determine which buffers should be played on which sources, by implementing an 'AttachBuffer' function. This function will perform the duplication and resource acquisition operations, and apply all the 3D Audio and EAX Source parameters that have been set on that Source, to the newly created buffer. The DirectSound EAX4Demo application in the EAX 4.0 SDK uses the method described here.

EAX Set and Get Functionality

In both Direct Sound and Open AL, all EAX parameters are controlled using two functions / methods. There is a 'set' call for applying new parameter values, and a 'get' call for retrieving the current values. The calls use similar parameter types; a GUID and parameter enumeration are used to identify which parameter of which EAX Object is being addressed, and a void pointer and data size are used to pass the data to / from the EAX engine.

GUID Type

A parameter of type GUID (Global Unique IDentifier) will identify the EAX Object that is being referenced. This will typically be one of the following: -

EAXPROPERTYID_EAX40_Context
The property being set or get is a member property of the EAX 4.0 Context object

EAXPROPERTYID_EAX40_FXSlot*n* (where *n* is between 0 and 3)
The property being set or get is a member property of the EAX 4.0 FX Slot object, or an effect property consistent with the effect type loaded in slot *n*

EAXPROPERTYID_EAX40_Source
The property being set or get is a member property of the EAX 4.0 Source object.

Property Enumeration Value

EAX 4.0 enumerates each member property of each object. For example, the Context Object properties are numbered from EAXCONTEXT_NONE to EAXCONTEXT_LASTERROR. Pass in the correct property enumeration value to identify the parameter that should be altered or queried.

Data Pointer

A void * pointer that points to the location in memory where data will either be read from, in the case of a 'Set' call, or written to, in the case of a 'Get' call.

Data Size

An unsigned integer indicating the size of the data pointed to by the data pointer. If the data size is not big enough to store enough data for the parameter, the EAX call will fail.

Initializing EAX 4.0

The first step to implementing EAX effects is to determine if EAX support is available on the user's system. Next, it is useful to create high-level 'Set' and 'Get' functions, which pass parameters to and from each EAX object. These functions will be implemented for Direct Sound and Open AL, and will be used in all the following examples. All source code examples are taken from the EAX4Demo Visual C++ Workspaces found in the EAX 4.0 SDK. These are complete, build-able applications, and as such, they contain a number of functions that are not reproduced here in this guide for simplicity.

Open AL

After initializing OpenAL (by successfully creating a Device and Context), EAX can be initialized in two simple steps. Firstly, it is necessary to determine if EAX is support is available, and secondly, if it is available, the application needs to retrieve pointers to the two EAX function calls. The prototypes for the Open AL EAX extension functions are: -

```
typedef ALenum (*EAXSet)(const GUID*, ALuint, ALuint,
      ALvoid*, ALuint);
typedef ALenum (*EAXGet)(const GUID*, ALuint, ALuint,
      ALvoid*, ALuint);
```

Each function takes a pointer to the EAX Object (GUID), the enumeration value of the property in the object, the name of an AL Source, a pointer to the data, and the size of the data pointed to, respectively.

To determine if EAX is present, the application should use the OpenAL function **alIsExtensionPresent**. If the extension is found, then the Open AL function call **alGetProcAddress** can be used to get the pointers to the two EAX extension functions.

```
// Global variables

bool g_bInitialized = false;
bool g_bEAXInitialized = false;

unsigned long g_ulNumSources = 0;

ALuint g_SourceNames[MAX_SOURCES] = { 0 };


// EAX Function pointers

EAXSet      g_EAXSet = NULL;
EAXGet      g_EAXGet = NULL;


void InitEAX4()
{
  if (alIsExtensionPresent((ALubyte*)"EAX4.0"))
  {
    // Obtain the pointers to the EAX Functions
    g_EAXSet =
      (EAXSet)alGetProcAddress((ALubyte*)"EAXSet");
    g_EAXGet =
      (EAXGet)alGetProcAddress((ALubyte*)"EAXGet");

    if ((g_EAXSet) && (g_EAXGet))
    {
      g_bEAXInitialized = true;
    }
  }
}
```

It is not necessary to perform any clean up of the EAX extension in OpenAL, so a shutdown function is trivial: -

```
void ShutdownEAX4()
{
  g_EAXSet = NULL;
  g_EAXGet = NULL;

  g_bEAXInitialized = false;
}
```

EAX calls can be divided into two types – those that only affect a single Source, and those that can affect all sources. It can be helpful to write wrapper functions for these two types of call, especially if writing an audio library that will support both Open AL and Direct Sound. The Set and Get global properties function calls do not need a Source name because the Open AL library will take care of passing the call to the driver.

```
bool SetGlobalEAXProperty(GUID guid, unsigned long
      ulProperty, void *pData, unsigned long ulDataSize)
{
  bool bResult = false;

  if (g_bEAXInitialized)
  {
    if (g_EAXSet(&guid, ulProperty, 0, pData, ulDataSize)
      == AL_NO_ERROR)
      bResult = true;
  }

  return bResult;
}


bool GetGlobalEAXProperty(GUID guid, unsigned long
      ulProperty, void *pData, unsigned long ulDataSize)
{
  bool bResult = false;

  if (g_bEAXInitialized)
  {
    if (g_EAXGet(&guid, ulProperty, 0, pData, ulDataSize)
      == AL_NO_ERROR)
      bResult = true;
  }

  return bResult;
}
```

The function calls used to Set and Get source-specific properties need to be passed a Source identifier because they concern an effect that will apply to a Source. In the following example code, the SourceIndex parameter is simply an index into an array of sources that were created by the audio engine at initialization time.

```
bool SetSourceEAXProperty(SOURCEINDEX SourceIndex, GUID
      guid, unsigned long ulProperty, void *pData,
      unsigned long ulDataSize)
{
  bool bResult = false;

  if ((g_bInitialized) && (g_bEAXInitialized))
  {
    if ((SourceIndex < g_ulNumSources) &&
      (g_SourceNames[SourceIndex]))
    {
      if (g_EAXSet(&guid, ulProperty,
      g_SourceNames[SourceIndex], pData, ulDataSize) ==
      AL_NO_ERROR)
        bResult = true;
    }
  }

  return bResult;
}
```

```
bool GetSourceEAXProperty(SOURCEINDEX SourceIndex, GUID
      guid, unsigned long ulProperty, void *pData,
      unsigned long ulDataSize)
{
  bool bResult = false;

  if ((g_bInitialized) && (g_bEAXInitialized))
  {
    if ((SourceIndex < g_ulNumSources) &&
      (g_SourceNames[SourceIndex]))
    {
      if (g_EAXGet(&guid, ulProperty,
      g_SourceNames[SourceIndex], pData, ulDataSize) ==
      AL_NO_ERROR)
        bResult = true;
    }
  }

  return bResult;
}
```

Finally, it is worth noting that Open AL does not currently include a function call to adjust the scale of the units used by all distance related parameters (such as position, and reference distance). EAX assumes that units are in meters, so it is important either to multiply all distance parameters by a scale value before passing them to OpenAL, or to use the EAX 4.0 Context Distance Factor property. For more information about Distance Units, please refer to the section on *Distance Units*.

Direct Sound

EAX is exposed in DirectSound as a property set extension. Property sets were introduced in Direct X 5.0 and provide a way for manufacturers to expose properties that are not native to DirectSound. Creative uses this interface to add EAX properties to DirectSound objects.

After Direct Sound has been initialized, to determine if there is support for EAX it is necessary to obtain a pointer to a Property Set interface. The Property Set interface must be queried from a Direct Sound 3D Buffer interface that has been queried from a Direct Sound secondary buffer created in hardware, with 3D controls. It is a good idea to hold on to this secondary buffer and all its interfaces for the life of the application, because this buffer can be used to apply global EAX properties.

Once the Property Set interface is acquired, a call to the QuerySupport method can be used to check for support for a particular property of an object. A strict application could check for support for every property it wishes to use. However, it is normally sufficient to simply check for support for the 'All Parameters' property of each object the application wishes to use. The following helper function checks for support of the given object (GUID) and property.

```
bool TestSupport(LPKSPROPERTYSET lpPropSet, GUID Guid,
      unsigned long ulProperty)
{
  unsigned long ulSupport;
  bool bReturn = false;
```

```
  if (lpPropSet)
  {
    if (SUCCEEDED(lpPropSet->QuerySupport(Guid,
      ulProperty, &ulSupport)))
    {
      if ((ulSupport & (KSPROPERTY_SUPPORT_GET |
      KSPROPERTY_SUPPORT_SET)) == (KSPROPERTY_SUPPORT_GET
      | KSPROPERTY_SUPPORT_SET))
      {
        bReturn = true;
      }
    }
  }

  return bReturn;
}
```

The following code shows how to initialize EAX in Direct Sound, utilizing the **TestSupport** function defined above. The dummy buffer created will be used to query for support of EAX, and to set global EAX properties, so the interfaces are stored in global variables. The dummy buffer must be created in hardware (it is not sufficient to use the DSBCAPS_LOCDEFER flag), and must have 3D controls (DSBCAPS_CTRL3D). It is also important that the buffer size is an exact multiple of the wave format block alignment or else the call to create the buffer will fail.

```
// Direct Sound Object

LPDIRECTSOUND8                g_lpDS8 = NULL;
LPDIRECTSOUNDBUFFER           g_lpDSBPrimary = NULL;
LPDIRECTSOUND3DLISTENER       g_lpDS3DListener = NULL;

// Direct Sound Secondary Buffer and interfaces (for
      global EAX control)

LPDIRECTSOUNDBUFFER           g_lpEAXDSB = NULL;
LPDIRECTSOUNDBUFFER8          g_lpEAXDSB8 = NULL;
LPDIRECTSOUND3DBUFFER8        g_lpEAXDS3DB8 = NULL;
LPKSPROPERTYSET               g_lpEAXPropSet = NULL;

bool        g_bInitialized = false;
bool        g_bEAXInitialized = false;
unsigned long     g_ulNumSources = 0;
unsigned long     g_ulNumBuffers = 0;
SOURCE            g_Sources[MAX_SOURCES];
BUFFER            g_Buffers[MAX_BUFFERS];

void InitEAX4()
{
  DSBUFFERDESC dsbdSecondary;
  WAVEFORMATEXTENSIBLE  wfext;
  unsigned long ulDatasize;

  // Creating dummy DS buffer
  ulDatasize = 64;
  wfext.Format.wFormatTag = 1;
  wfext.Format.nChannels = 1;
  wfext.Format.wBitsPerSample = 16;
```

```
wfext.Format.nSamplesPerSec = 22050;
wfext.Format.nBlockAlign = (wfext.Format.wBitsPerSample
    / 8) * wfext.Format.nChannels;
wfext.Format.nAvgBytesPerSec =
    wfext.Format.nSamplesPerSec *
    wfext.Format.nBlockAlign;
wfext.Format.cbSize = 0;

// Fill in our DS Secondary buffer description
memset(&dsbdSecondary, 0, sizeof(DSBUFFERDESC));
dsbdSecondary.dwSize = sizeof(DSBUFFERDESC);
dsbdSecondary.dwBufferBytes = ulDatasize;
dsbdSecondary.dwFlags = DSBCAPS_LOCHARDWARE |
    DSBCAPS_CTRL3D;
dsbdSecondary.lpwfxFormat = (LPWAVEFORMATEX)&wfext;

if(SUCCEEDED(g_lpDS8->CreateSoundBuffer(&dsbdSecondary,
    &g_lpEAXDSB, NULL)))
{
  if (SUCCEEDED(g_lpEAXDSB-
    >QueryInterface(IID_IDirectSoundBuffer8,
    (void**)&g_lpEAXDSB8)))
  {
    if (SUCCEEDED(g_lpEAXDSB8->
    QueryInterface(IID_IDirectSound3DBuffer8,
    (void**)&g_lpEAXDS3DB8)))
    {
      if (SUCCEEDED(g_lpEAXDS3DB8->
    QueryInterface(IID_IKsPropertySet,
    (void**)&g_lpEAXPropSet)))
      {
        // Determine if we have EAX 4.0 Support

        g_bEAXInitialized = true;

        g_bEAXInitialized |=
    TestSupport(g_lpEAXPropSet,
    EAXPROPERTYID_EAX40_Context,
    EAXCONTEXT_ALLPARAMETERS);

        g_bEAXInitialized |=
    TestSupport(g_lpEAXPropSet,
    EAXPROPERTYID_EAX40_FXSlot0,
    EAXFXSLOT_ALLPARAMETERS);

        g_bEAXInitialized |=
    TestSupport(g_lpEAXPropSet,
    EAXPROPERTYID_EAX40_FXSlot1,
    EAXFXSLOT_ALLPARAMETERS);

        g_bEAXInitialized |=
    TestSupport(g_lpEAXPropSet,
    EAXPROPERTYID_EAX40_FXSlot2,
    EAXFXSLOT_ALLPARAMETERS);
```

```
            g_bEAXInitialized |=
          TestSupport(g_lpEAXPropSet,
          EAXPROPERTYID_EAX40_FXSlot3,
          EAXFXSLOT_ALLPARAMETERS);

            g_bEAXInitialized |=
          TestSupport(g_lpEAXPropSet,
          EAXPROPERTYID_EAX40_Source,
          EAXSOURCE_ALLPARAMETERS);
            }
          }
        }
      }

      if (!g_bEAXInitialized)
        ShutdownEAX4();
    }
```

When the application quits, it should release the dummy buffer and interfaces created in the **InitEAX4** function.

```
    void ShutdownEAX4()
    {
      if (g_lpEAXPropSet)
      {
        g_lpEAXPropSet->Release();
        g_lpEAXPropSet = NULL;
      }

      if (g_lpEAXDS3DB8)
      {
        g_lpEAXDS3DB8->Release();
        g_lpEAXDS3DB8 = NULL;
      }

      if (g_lpEAXDSB8)
      {
        g_lpEAXDSB8->Release();
        g_lpEAXDSB8 = NULL;
      }

      if (g_lpEAXDSB)
      {
        g_lpEAXDSB->Release();
        g_lpEAXDSB = NULL;
      }

      g_bEAXInitialized = false;
    }
```

EAX calls can be divided into two types – those that only affect a single Source, and those that can affect all sources. It can be helpful to write wrapper functions for these two types of call, especially if writing an audio library that will support both Open AL and Direct Sound. The Set and Get global properties function calls will be made using the property set interface obtained from the dummy buffer created above.

```
bool SetGlobalEAXProperty(GUID guid, unsigned long
      ulProperty, void *pData, unsigned long ulDataSize)
{
  bool bResult = false;

  if (g_bEAXInitialized)
  {
    if (SUCCEEDED(g_lpEAXPropSet->Set(guid, ulProperty,
      NULL, 0, pData, ulDataSize)))
      bResult = true;
  }

  return bResult;
}


bool GetGlobalEAXProperty(GUID guid, unsigned long
      ulProperty, void *pData, unsigned long ulDataSize)
{
  unsigned long ulBytesReturned;
  bool bResult = false;

  if (g_bEAXInitialized)
  {
    if (SUCCEEDED(g_lpEAXPropSet->Get(guid, ulProperty,
      NULL, 0, pData, ulDataSize, &ulBytesReturned)))
      bResult = true;
  }

  return bResult;
}
```

The Set and Get Source properties functions will need to be made using the Property Set interface obtained from the appropriate Direct Sound Buffer. The SourceIndex parameter is an index into an array of Source structures that store the various pointers to the Direct Sound Buffer and all its interfaces (including the IksPropertySet interface). Note that the SetSourceEAXProperty function includes a call to StoreSourceProperty. This functionality is necessary with this audio implementation because the audio code assumes that any settings made on a Source will affect any Buffer that is / will become attached (see the *Direct Sound Voice Management* section). The 3D audio and EAX settings are therefore specific to the Source not the buffer. When a different buffer is attached to the Source, the 3D and EAX settings stored for that Source are immediately applied to the duplicated Direct Sound Buffer (see the AttachBuffer function in DSAudio.cpp in the SDK for more details).

```
bool SetSourceEAXProperty(SOURCEINDEX SourceIndex, GUID
      guid, unsigned long ulProperty, void *pData,
      unsigned long ulDataSize)
{
  bool bResult = false;

  if ((g_bInitialized) && (g_bEAXInitialized))
  {
    if (SourceIndex < g_ulNumSources)
    {
      // Store Source Property
```

```
        StoreSourceProperty(SourceIndex, ulProperty, pData,
        ulDataSize);

        // If a Source exits, then apply EAX property
        if (g_Sources[SourceIndex].lpPS)
        {
          if (SUCCEEDED(g_Sources[SourceIndex].lpPS->Set(
        guid, ulProperty, NULL, 0, pData, ulDataSize)))
          bResult = true;
        }
      }
    }

    return bResult;
  }


  bool GetSourceEAXProperty(SOURCEINDEX SourceIndex, GUID
        guid, unsigned long ulProperty, void *pData,
        unsigned long ulDataSize)
  {
    unsigned long ulBytesReturned;
    bool bResult = false;

    if ((g_bInitialized) && (g_bEAXInitialized))
    {
      if ((SourceIndex < g_ulNumSources) &&
        (g_Sources[SourceIndex].lpPS))
      {
        if (SUCCEEDED(g_Sources[SourceIndex].lpPS-
        >Get(guid, ulProperty, NULL, 0, pData, ulDataSize,
        &ulBytesReturned)))
          bResult = true;
      }
    }

    return bResult;
  }
```

A final point to note about initialization is that EAX assumes that all units are in meters, so it is important either to multiply all distance parameters by a scale value before passing them to Direct Sound, or to use the SetDistanceFactor method of the Direct Sound 3D Listener Interface to perform this conversion. For more information about Distance Units, please refer to the *Distance Units* section.

Controlling the Context

The following examples use the SetGlobalEAXProperty and GetGlobalEAXProperty functions defined above.

Setting PrimaryFXSlotID

```
// Set the Context Primary FX Slot ID to FX Slot 0

GUID PrimaryGuid;
```

```
memcpy(&PrimaryGuid, &EAXPROPERTYID_EAX40_FXSlot0,
      sizeof(GUID));

SetGlobalEAXProperty(EAXPROPERTYID_EAX40_Context,
      EAXCONTEXT_PRIMARYFXSLOTID, &PrimaryGuid,
      sizeof(GUID));
```

Getting PrimaryFXSlotID

```
GUID GetPrimaryGuid;

GetGlobalEAXProperty(EAXPROPERTYID_EAX40_Context,
      EAXCONTEXT_PRIMARYFXSLOTID, &GetPrimaryGuid,
      sizeof(GUID));
```

Setting Distance Factor

```
float flDistanceFactor;

// Assume distance units are in feet, so one unit ==
      0.3048 metres.
flDistanceFactor = 0.3048f;

SetGlobalEAXProperty(EAXPROPERTYID_EAX40_Context,
      EAXCONTEXT_DISTANCEFACTOR, &flDistanceFactor,
      sizeof(float));
```

Getting Distance Factor

```
float flGetDistanceFactor;

GetGlobalEAXProperty(EAXPROPERTYID_EAX40_Context,
      EAXCONTEXT_DISTANCEFACTOR, &flGetDistanceFactor,
      sizeof(float));
```

Controlling FX Slots

The following examples use the SetGlobalEAXProperty and GetGlobalEAXProperty functions defined above.

Setting FX Slot Lock state

```
// Locking FX Slot 2
long lLock;

lLock = EAXFXSLOT_LOCKED;
```

```
SetGlobalEAXProperty(EAXPROPERTYID_EAX40_FXSlot2,
      EAXFXSLOT_LOCK, &lLock, sizeof(long));
```

Getting FX Slot Lock state

```
long lGetLock;

GetGlobalEAXProperty(EAXPROPERTYID_EAX40_FXSlot2,
      EAXFXSLOT_LOCK, &lGetLock, sizeof(long));
```

Setting FX Slot Effect

```
// Loading an Autowah into Slot 3

SetGlobalEAXProperty(EAXPROPERTYID_EAX40_FXSlot3,
      EAXFXSLOT_LOADEFFECT, (void*)&EAX_AUTOWAH_EFFECT,
      sizeof(GUID));
```

Getting FX Slot Effect

```
GUID Effect;

GetGlobalEAXProperty(EAXPROPERTYID_EAX40_FXSlot3,
      EAXFXSLOT_LOADEFFECT, &Effect, sizeof(GUID));
```

Setting FX Slot Configuration

If an application wishes to configure an FX Slot in one call, it can use the EAXFXSLOT_ALLPARAMETERS enumeration: -

```
// Configuring an FX Slot
EAXFXSLOTPROPERTIES FXSlot;

FXSlot.guidLoadEffect = EAX_ECHO_EFFECT;
FXSlot.lLock = EAXFXSLOT_LOCKED;
FXSlot.lVolume = 0;
FXSlot.ulFlags = 0;

SetGlobalEAXProperty(EAXPROPERTYID_EAX40_FXSlot2,
      EAXFXSLOT_ALLPARAMETERS, &FXSlot,
      sizeof(EAXFXSLOTPROPERTIES));
```

Getting FX Slot Configuration

The EAXFXSLOT_ALLPARAMETERS enumeration value can be used to retrieve all the settings of an FX Slot: -

```
EAXFXSLOTPROPERTIES GetFXSlot;

GetGlobalEAXProperty(EAXPROPERTYID_EAX40_FXSlot2,
      EAXFXSLOT_ALLPARAMETERS, &GetFXSlot,
      sizeof(EAXFXSLOTPROPERTIES));
```

Controlling Effect Parameters

The following examples use the SetGlobalEAXProperty and GetGlobalEAXProperty functions defined above.

Setting Effect Parameters

```
// Setting FX Slot 3 Autowah Attack Time
float flAttackTime;
flAttackTime = 0.5f;

SetGlobalEAXProperty(EAXPROPERTYID_EAX40_FXSlot3,
      EAXAUTOWAH_ATTACKTIME, &flAttackTime,
      sizeof(float));

// Setting FX Slot 2 Echo Damping

float flDamping;
flDamping = 0.3f;

SetGlobalEAXProperty(EAXPROPERTYID_EAX40_FXSlot2,
      EAXECHO_DAMPING, &flDamping, sizeof(float));



// Setting FX Slot 1 Chorus parameters to defaults

EAXCHORUSPROPERTIES eaxChorus;

eaxChorus.ulWaveform = EAXCHORUS_DEFAULTWAVEFORM;
eaxChorus.lPhase = EAXCHORUS_DEFAULTPHASE;
eaxChorus.flRate = EAXCHORUS_DEFAULTRATE;
eaxChorus.flDepth = EAXCHORUS_DEFAULTDEPTH;
eaxChorus.flFeedback = EAXCHORUS_DEFAULTFEEDBACK;
eaxChorus.flDelay = EAXCHORUS_DEFAULTDELAY;

SetGlobalEAXProperty(EAXPROPERTYID_EAX40_FXSlot1,
      EAXCHORUS_ALLPARAMETERS, &eaxChorus,
      sizeof(EAXCHORUSPROPERTIES));
```

Getting Effect Parameters

```
// Getting FX Slot 3 Autowah Attack Time

float flGetAttackTime;

GetGlobalEAXProperty(EAXPROPERTYID_EAX40_FXSlot3,
      EAXAUTOWAH_ATTACKTIME, &flGetAttackTime,
      sizeof(float));


// Getting FX Slot 2 Echo Damping
```

```
float flGetDamping;

GetGlobalEAXProperty(EAXPROPERTYID_EAX40_FXSlot2,
      EAXECHO_DAMPING, &flGetDamping, sizeof(float));


// Getting FX Slot 1 Chorus parameters

EAXCHORUSPROPERTIES eaxGetChorus;

GetGlobalEAXProperty(EAXPROPERTYID_EAX40_FXSlot1,
      EAXCHORUS_ALLPARAMETERS, &eaxGetChorus,
      sizeof(EAXCHORUSPROPERTIES));
```

Controlling Source Properties

The following examples use the SetSourceEAXProperty and GetSourceEAXProperty functions defined above.

Setting Obstruction

```
// Applying an Obstruction preset

EAXOBSTRUCTIONPROPERTIES eaxOB;

eaxOB.lObstruction = -1000;
eaxOB.flObstructionLFRatio = 0.25f;

SetSourceEAXProperty(0, EAXPROPERTYID_EAX40_Source,
      EAXSOURCE_OBSTRUCTIONPARAMETERS, &eaxOB,
      sizeof(EAXOBSTRUCTIONPROPERTIES));
```

Getting Obstruction

```
EAXOBSTRUCTIONPROPERTIES eaxGetOB;

GetSourceEAXProperty(0, EAXPROPERTYID_EAX40_Source,
      EAXSOURCE_OBSTRUCTIONPARAMETERS, &eaxGetOB,
      sizeof(EAXOBSTRUCTIONPROPERTIES));
```

Setting Active FX Slots

```
// Enable Sends from Source 0 to FX Slot 2 and FX Slot 3
EAXACTIVEFXSLOTS eaxActive;

memcpy(&eaxActive.guidActiveFXSlots[0],
      &EAXPROPERTYID_EAX40_FXSlot2, sizeof(GUID));

memcpy(&eaxActive.guidActiveFXSlots[1],
      &EAXPROPERTYID_EAX40_FXSlot3, sizeof(GUID));
```

```
SetSourceEAXProperty(0, EAXPROPERTYID_EAX40_Source,
      EAXSOURCE_ACTIVEFXSLOTID, &eaxActive,
      sizeof(EAXACTIVEFXSLOTS));
```

Getting Active FX Slots

```
EAXACTIVEFXSLOTS eaxGetActive;

GetSourceEAXProperty(0, EAXPROPERTYID_EAX40_Source,
      EAXSOURCE_ACTIVEFXSLOTID, &eaxGetActive,
      sizeof(EAXACTIVEFXSLOTS));
```

Setting Send Levels

The EAX Source send levels call can accept one or more EAXSOURCESENDPROPERTIES structures. In this example two structures are used to set-up the Send levels to two different FX Slots.

```
// Setting Send levels to Slots 2 and 3
EAXSOURCESENDPROPERTIES eaxSend[2];

eaxSend[0].guidReceivingFXSlotID =
      EAXPROPERTYID_EAX40_FXSlot2;
eaxSend[0].lSend = -1000;
eaxSend[0].lSendHF = 0;

eaxSend[1].guidReceivingFXSlotID =
      EAXPROPERTYID_EAX40_FXSlot3;
eaxSend[1].lSend = 0;
eaxSend[1].lSendHF = -1000;

SetSourceEAXProperty(0, EAXPROPERTYID_EAX40_Source,
      EAXSOURCE_SENDPARAMETERS, &eaxSend[0],
      sizeof(EAXSOURCESENDPROPERTIES) * 2);
```

Getting Send Levels

There is no way to request the Send levels for a particular FX Slot; instead, the application must retrieve the Source Sends levels to all FX Slots. The application can then search through the array of returned EAXSOURCESENDPROPERTIES structures to find the structure containing the Send levels to the desired FX Slot (by comparing the guidReceivingFXSlotID member of the structure to appropriate FX Slot guid). The eax.h file contains a #define called EAX_MAX_FXSLOTS that is set to the number of FX Slots currently defined (4). It is a good idea to use this definition if the application wishes to retrieve the Send levels to any FX Slot.

```
EAXSOURCESENDPROPERTIES eaxGetSend[EAX_MAX_FXSLOTS];
unsigned long ulLoop;
```

```
GetSourceEAXProperty(0, EAXPROPERTYID_EAX40_Source,
      EAXSOURCE_SENDPARAMETERS, &eaxGetSend[0],
      sizeof(EAXSOURCESENDPROPERTIES) * EAX_MAX_FXSLOTS);

// Display Send levels from Source to FX Slot 2 and 3

for (ulLoop = 0; ulLoop < EAX_MAX_FXSLOTS; ulLoop++)
{
  if (!memcmp(&EAXPROPERTYID_EAX40_FXSlot2,
      &eaxGetSend[ulLoop].guidReceivingFXSlotID,
      sizeof(GUID)))
  {
    printf("FX Sends to Slot 2 - Send %d : Send HF %d\n",
      eaxGetSend[ulLoop].lSend,
      eaxGetSend[ulLoop].lSendHF);
  }
  else if (!memcmp(&EAXPROPERTYID_EAX40_FXSlot3,
      &eaxGetSend[ulLoop].guidReceivingFXSlotID,
      sizeof(GUID)))
  {
    printf("FX Sends to Slot 3 - Send %d : Send HF %d\n",
      eaxGetSend[ulLoop].lSend,
      eaxGetSend[ulLoop].lSendHF);
  }
}
```

Setting All Send Levels

Like the EAXSOURCE_SENDPARAMETERS parameter, the EAXSOURCE_ALLSENDPARAMETERS parameter can accept one or more structures. In this case, the call expects EAXSOURCEALLSENDPROPERTIES structures. In this example, only one structure is used to configure all the Sends to FX Slot 0.

```
// Setting all Send levels from Source to Slot 0

EAXSOURCEALLSENDPROPERTIES eaxAllSend;

eaxAllSend.guidReceivingFXSlotID =
      EAXPROPERTYID_EAX40_FXSlot0;
eaxAllSend.lSend = -500;
eaxAllSend.lSendHF = 0;
eaxAllSend.lOcclusion = -1000;
eaxAllSend.flOcclusionLFRatio = 0.25f;
eaxAllSend.flOcclusionRoomRatio = 1.0f;
eaxAllSend.flOcclusionDirectRatio = 1.5f;
eaxAllSend.lExclusion = -2000;
eaxAllSend.flExclusionLFRatio = 0.5f;

SetSourceEAXProperty(0, EAXPROPERTYID_EAX40_Source,
      EAXSOURCE_ALLSENDPARAMETERS, &eaxAllSend,
      sizeof(EAXSOURCEALLSENDPROPERTIES));
```

Getting All Send Levels

There is no way to request all the send levels for a particular FX Slot; instead, the application must retrieve all the Source sends levels to all of the FX Slots. The application can then search through the array of returned EAXSOURCEALLSENDPROPERTIES structures to find the structure containing all the send levels to the desired FX Slot (by comparing the guidReceivingFXSlotID member of the structure to the appropriate FX Slot guid). The eax.h file contains a #define called EAX_MAX_FXSLOTS that is set to the number of FX Slots currently defined (4). It is a good idea to use this definition if the application wishes to retrieve the Send levels to any FX Slot.

```
EAXSOURCEALLSENDPROPERTIES
      eaxGetAllSend[EAX_MAX_FXSLOTS];
unsigned long ulLoop;

GetSourceEAXProperty(0, EAXPROPERTYID_EAX40_Source,
      EAXSOURCE_ALLSENDPARAMETERS, &eaxGetAllSend,
      sizeof(EAXSOURCEALLSENDPROPERTIES) *
      EAX_MAX_FXSLOTS);

// Display (some of the) send levels to FX Slot 0
for (ulLoop = 0; ulLoop < EAX_MAX_FXSLOTS; ulLoop++)
{
  if (!memcmp(&EAXPROPERTYID_EAX40_FXSlot0,
      &eaxGetAllSend[ulLoop].guidReceivingFXSlotID,
      sizeof(GUID)))
  {
    printf("FX Sends to Slot 0 - Send %d : Send HF %d :
      Occlusion %d\n", eaxGetAllSend[ulLoop].lSend,
      eaxGetAllSend[ulLoop].lSendHF,
      eaxGetAllSend[ulLoop].lOcclusion);
    break;
  }
}
```

Error Checking

A newly introduced feature for EAX 4.0 is the ability to get an EAX specific error code in the event that an EAX call does not complete successfully. If an EAX 4.0 call fails, then the EAX Error code is set to one of the values listed below to indicate the reason for the failure. A successful EAX 4.0 call does not set the EAX Error code to EAX_OK, so the Error code always indicates why the last call that failed, failed. The Error code is automatically reset (to EAX_OK) after the application does a 'get'. It is not possible for an application to 'set' the error code. The following error codes have been defined:

EAX 4.0 Error Codes

EAX_OK
No EAX calls have failed since the last call to get the EAX Error code.

EAXERR_INVALID_OPERATION
The last failed EAX call was illegal – for example, an attempt was made to change the Effect type in an FX Slot that was locked by another application.

EAXERR_INVALID_VALUE
The last failed EAX call made an attempt to set an EAX parameter to an out-of-range value, or the size of the data pointed to was not large enough to contain the expected data.

EAXERR_NO_EFFECT_LOADED
The last failed EAX call attempted to adjust an effect parameter on an FX Slot that did not contain an Effect.

EAXERR_UNKNOWN_EFFECT
The last failed EAX call attempted to load an unrecognized Effect type.

Getting EAX Error Code

The following example attempts to load a bad effect type into FX Slot 2. The **GetEAXErrorString** function is a helper function defined in DSAudio.cpp as well as in ALAudio.cpp that returns a string name and explanation for the EAX Error code returned.

```
GUID BadGuid;
long lEAXError;
char szString[256];

memset(&BadGuid, 0xFF, sizeof(GUID));

// This call is expected to fail

if (!SetGlobalEAXProperty(EAXPROPERTYID_EAX40_FXSlot2,
      EAXFXSLOT_LOADEFFECT, &BadGuid, sizeof(GUID)))
{
  // Check EAX Error Code
  GetGlobalEAXProperty(EAXPROPERTYID_EAX40_Context,
      EAXCONTEXT_LASTERROR, &lEAXError, sizeof(long));

  printf("EAX Error code was %d : %s\n", lEAXError,
      GetEAXErrorString(lEAXError, szString,
      sizeof(szString)));
}
```

Environmental Audio Programming Techniques

The previous section, *Programming the EAX 4.0 interface*, dealt with the nitty-gritty of setting up and managing the EAX engine using EAX's interface objects to pass values to and from the hardware. This section looks at EAX from a higher-level perspective – from the point of view of integrating EAX into the rest of the application.

It is important that you set up EAX in such a way that your application can co-exist happily with other applications, which might also seek to share the EAX processing resources.

In applications that include the notion of a simulated 3D world, it is also crucial that the distance model used to inform the 3D audio API how sounds are spaced in the world is compatible with EAX's own distance model.

In an application scenario, the program logic and interaction with the user should provide the audio engine with sufficient information to adjust the configuration of EAX effects in real-time. For an EAX 4.0 application with multiple environment support, the mechanisms needed to maintain a lifelike acoustic simulation are not trivial.

This section covers all these aspects of EAX application design, giving you the knowledge to make sure your EAX 4.0 application is robust and efficient, and most importantly sounds great! The following information is not necessarily a recipe for the perfect EAX 4.0 audio implementation. Rather it is a set of suggestions, inspired by Creative's experiences working with a large number of commercial 3D audio projects.

Distance Units

As mentioned previously, EAX is very closely integrated with the 3D Audio API being used (Open AL or Direct Sound 3D). It is therefore essential to understand that the units of distance used in the Audio API will have an impact on the equations used by the EAX engine.

If a 3D Audio application does not use Doppler effects or EAX, then it is not critical for the audio API to know the scale of the units being used. The Direct Sound 3D Distance Model and the Open AL Inverse Distance Clamped model both use the following equation:

Attenuation = -20 * log10 (1 + ((dist – min_dist) / min_dist) * ROF)

Where: -

'**dist**' is the distance between Source and Listener

'**min_dist**' is the Direct Sound 3D Min Distance or the Open AL Reference Distance

'**ROF**' is the Roll-Off Factor

Therefore, a Source with a roll-off factor of 1, a Min Distance of 5 units, at a distance of 10 units, has: -

Attenuation = -20 * log10 (1 + ((10 – 5) / 5) * 1)

= -6.02 dB

If we multiply all distance units by 1000, to get a Source with a Min Distance of 5000 units, at a distance of 10000 units, the attenuation is: -

Attenuation = -20 * log10 (1 + ((10000 – 5000) / 5000) * 1)

= -6.02 dB

In both scenarios, the source will drop to half volume when at a distance of twice the Min Distance – regardless of the scale of the units.

However, if EAX support is added to the application, the EAX engine expects units to be in meters so that it can apply realistic fall-off of both the original sound (direct path) and the reverb (wet path). Distance related parameters are used in the calculation of Air Absorption and the Reverb statistical model.

There is a big difference between the amount of air absorption in 5 meters of air, versus 5 kilometers of air! It is therefore critical to ensure that the units of distance used by the application are converted into meters. In Direct Sound 3D, this can be done using the **SetDistanceFactor** method of the Direct Sound 3D Listener Interface. In Open AL, there is no equivalent call to adjust the distance factor, so an application should use the **EAX 4.0 Context Distance Factor** call instead.

By setting the Distance Factor to 0.001 in the second example, the EAX engine will apply the appropriate amount of roll-off to the direct and wet paths.

If the application is unable to adjust the Distance Factor – then the application has two choices. It can manually multiply all distance related parameters by a scale factor before passing them to the audio engine (this includes positions, velocities, and the min and max distance for each Source). Alternatively, the application can disable the features of EAX that will be adversely affected by an incorrect distance scale, by doing the following: -

Ensure Source Air Absorption Factor is set to 0.0 (this is the default value).

Ensure Source Room Roll-off Factor is set to 0.0 (this is the default value).

Disable Source Room Auto and Room HF Auto Flags (both flags are enabled (set to true) by default).

Ensure (all) Reverb Room Roll-off Factors are set to 0.0 (this is the default value).

Note: By removing the Room Auto and Room HF Auto flags, no distance based attenuation will be applied to the reverb send level from a source. If a source is directional, then disabling these flags will also remove the additional attenuation applied to the reverb send level based on the source's directivity.

Creating a co-operative EAX application

The API has been designed with the intention that applications with support for EAX 4.0 should be able to happily co-exist in a multi-tasking environment. The hardware resources provided by EAX 4.0 hardware, while powerful, are not limitless. Therefore, an EAX 4.0 application has the ability to reserve resources for its own use by 'locking' FX Slots, preventing other applications from changing the effect types loaded in those slots.

When a slot is locked, only the locking application can load a new effect type in that slot, or unlock the slot. However, in order to maximize the sharing of resources as far

as possible, a locked slot is not totally inaccessible to another application. The parameters of a locked slot can still be adjusted by a subsequent application, and valid sound sources created in a subsequent program can still feed those slots.

Why does EAX 4.0 allow locking of the effect type loaded in an FX Slot, but not the effect's actual parameters? The process of changing effect parameters takes relatively few processor cycles – a millisecond at very worst case - so an application can rectify altered parameters quickly. On the other hand, loading a new effect type is a processor intensive procedure, and so frequent changing of effect types should be avoided to maintain smooth program operation.

Therefore its advisable that the first thing an application should do when it has successfully queried for EAX, is to check whether the slots it wants to use are already locked (see *Getting FX Slot Lock state*). If it is important that the effect type loaded remains under the control of the program, and the required slots are not locked, then the next steps are to load the desired effect type (see *Setting FX Slot Effect*) and lock the required FX Slots (see *Setting FX Slot Lock state*). Note that in order to preserve compatibility with legacy audio applications, FX Slots 0 and 1 are always flagged as locked on Creative Labs' current SoundBlaster Audigy drivers. Slot 0 always has EAX Reverb loaded, and slot 1 always has EAX Chorus loaded.

If an application detected that the FX Slots it required were already locked by another program, then there are a number of potential paths to take. If full EAX 4.0 support, including the ability to control the loaded effect type in each slot is critical, then the application should gracefully fail, displaying a message notifying the user that another application has locked the required EAX resources. However, it might be that the FX Slots, while locked, still contain the desired effect type. In this case, it could be acceptable to continue using the FX Slots in question and rely on the locking application not loading new effect types into the required slots. Moreover, the possibility exists that the application could continue to run but working with reduced or no EAX resources.

On Windows platforms, multimedia applications commonly scan for situations where the Windows focus is lost and then regained. If your application is using FX Slot resources, whether or not it locked the slots, you should consider caching the desired EAX effect parameters, and checking them against the parameters in the slots (using the "Get" functions, see Getting Effect Parameters) when this happens. Caching EAX parameters is desirable for performance reasons (see Solutions for), but also bear in mind that another application is capable of changing effect parameters in locked slots.

A 'co-operative' EAX 4.0 application should release any resources it has locked when it terminates. After an EAX 4.0 application terminates, the EAX Engine will restore the contents of the FX Slots and their locked status to the values they were set to before the application was launched. However, if the EAX 4.0 application exits prematurely, due to a crash, then the EAX Engine may not be aware that the application has exited. If the application locked an FX Slot, then that slot cannot be unlocked by any application, forcing a re-boot of the OS. It may be good practice therefore to disable FX Slot locking for Debug builds, or even make your application's locking behavior selectable with a configuration file switch.

Creating a multi-environment world with EAX 4.0

EAX 4.0's ability to render several audio effects at once allows the developer to simulate a world containing multiple environments, with the hardware FX Slots rendering different EAX Reverb effects for environments likely to be heard by the listener. Designing a mechanism to correctly update all the EAX 4.0 properties for an

accurate multi-environment simulation is not trivial; a number of tasks are involved. An environmental audio management layer must:

- Ensure that the listener's environment is always being correctly rendered.
- Ensure that the nearby environments most likely to be audible are being correctly rendered in other available FX Slots.
- Ensure that each sound source is feeding the correct two FX Slots.
- Ensure that each EAX reverb is correctly localized, to correspond with the location of its environment in the game world.

These are in addition to the environmental audio management tasks associated with a single-environment simulation. These include checking for / setting occlusion, exclusion and obstruction effects on each sound source.

The exact manner in which the user interaction, application logic and world geometry influence the audio engine is very much specific to each individual project. However, there is a general set of principles that will apply in most instances.

Environmental Zones

Correctly applying environmental audio involves establishing the interaction between the listener and sound sources in the virtual world, and the various environments that surround the listener and sources. The most common approach to managing environmental data is to separate the world's geometry into environmental zones. Each zone represents a separate acoustic enclosure in the world. The environmental audio management layer should be able to reference the co-ordinates for the listener and each sound source. Given any set of co-ordinates, it should be possible to uniquely identify the environmental zone in which the co-ordinates exist. Typically, a set of parameters for an EAX reverb effect will be associated with each unique zone identifier. This means that, for example, given co-ordinates showing the listener's position, it is simple to retrieve the set of EAX reverb parameters for the listener's zone.

Data structures such as binary search partition trees offer efficient solutions for storing and retrieving environmental zone data. The work of creating EAX Reverb parameter sets for each zone would commonly be undertaken by an audio designer, using an off-line tool.

The ability to separate world geometry into discrete zones, and reference a unique zone identifier from a given set of co-ordinates also helps when determining whether a sound should have occlusion effects applied. If it is detected that a source is in a different zone to the listener, then it is likely that some occlusion will be applied. The amount of occlusion applied when sound travels between two zones could be calculated dynamically at run-time. Again, this data could be prepared by an audio designer or pre-calculated algorithmically 'off-line' and stored in a look-up table.

Apertures between environmental zones

To work with more advanced environmental audio techniques such as exclusion and multiple environments, identifying 'apertures' or 'portals' between environmental zones is essential. An aperture can be defined as the area where a clear-air opening between two environmental zones exists, for example a doorway or window.

In a multiple environment system where three environments are being rendered, it is important that the two non-listener environments are correctly localized. If an aperture exists between the listener's zone and a secondary zone, then the secondary zone's reverb should be perceived to be emitting from the aperture. For information on localizing environmental reverb effects, see Reverb and Reflection Panning Algorithm.

Aperture data can also be used to add an extra degree of realism to occluded sounds. If a sound has been found to exist in an environment zone adjacent to the listener's, then the direct line between the sound source and the listener can be calculated. If this line intersects an aperture, then the sound should have exclusion applied instead of occlusion, because although the source and listener are in different environments the direct sound is not blocked.

Source to listener direct path

Efficiently determining whether a sound source should have an obstruction effect applied can be a tricky process. However, correctly used obstruction effects add a great deal to 3D audio in terms of realism. In a similar manner to exclusion, this can be achieved by calculating the direct line between the source and the listener, this time checking whether it intersects any world geometry.

A potential optimization is to assume that only a sound that is in the same environmental zone as the listener can possibly be obstructed. This pre-supposes that an occlusion effect applied to a sound from other environmental zones will mask any subsequent obstruction effect, not strictly a true assumption in the real world.

Pre-generating and storing axis-aligned bounding boxes that identify any potential obstruction (e.g. pillar, statue) within an environmental zone helps to increase efficiency greatly, but at the expense of some accuracy.

In real life, the diffraction of sound waves around an object means that the nearer the direct line between obstructed source and listener comes to the edge of the obstacle, the less filtering and attenuation takes place. So, to make obstruction effects sound even more natural, it is helpful to dynamically calculate the amount of obstruction to be applied according to the angle of incidence between the source, the edge of the obstruction, and the listener. Figure 3 shows how the amount of obstruction varies with the angle around the obstacle.

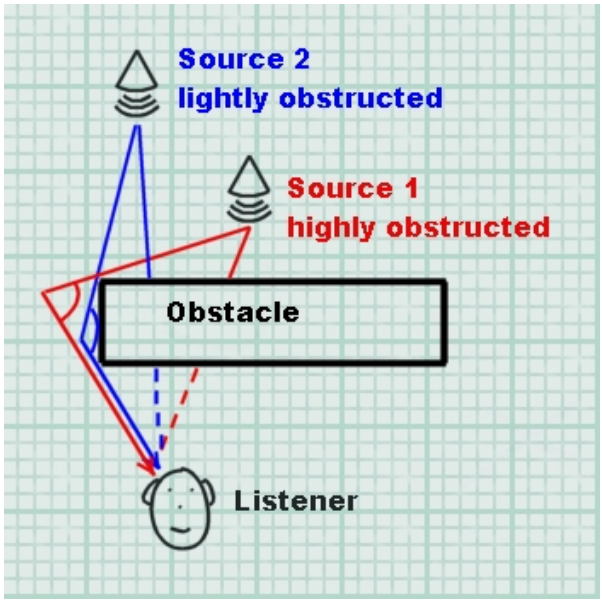

Figure 3 - Variable amounts of obstruction

Low-detail models and shared systems

A significant performance optimization for environmental data management is to utilize a low detail model of world geometry for audio calculations. As is often the case, the best efficiency gain comes when sharing geometry calculations with other application systems. For example, AI, collision-detection, and physics systems often involve low-detail geometrical models and similar mechanisms and calculations to those described above.

Multi-environment run-time management algorithm

Simulation of multiple environments is a key aim in the design of the EAX 4.0 SDK. Creative's engineers have proposed a general algorithm for environmental audio data management in a multiple-environment implementation. The following represents an example of how the EAX-related audio program flow might work for a game. In the following pseudo-code, '*n*' represents the maximum number of simultaneous environments that can be rendered (currently 3, which is EAX_MAX_FXSLOTS minus 1 because Slot 1 contains the Chorus effect).

Each audio frame:

Step 1: Update Environments

***If** listener position or orientation has changed **then**:*

Step 1a: Find the (*n*-1) closest environments to the listener (listener environment is always rendered):

- Option 1: Based on radius or priorities assigned to each environment zone
- Option 2: Based on distance from Listener to centre of environment
- Option 3: Based on environment's closest aperture

Step 1b: Update FX Slots

Mark the FX Slots that are NOT rendering one of the chosen 'n' environments selected above, as available and mute the FX Slot (EAXFXSLOT_VOLUME= -10000).

Traverse the list of n closest environments, and for every environment that is NOT currently being rendered, load it into the first available FX Slot (still muted), and mark that slot as unavailable.

Step 1c: Update Source Sends

For each Source:

Determine source's environment.

***If** an FX Slot is rendering this environment, **then** enable send (EAXSOURCE_ACTIVEFXSLOTID = FX Slot GUID of active environment, and EAX_PrimaryFXSlotID).*

***If** an FX Slot is not rendering this environment, **then** disable sends (EAXSOURCE_ACTIVEFXSLOTID = NULL, and EAX_PrimaryFXSlotID).*

***If** Source Environment == Listener Environment, **then** set Obstruction as required.*

***If** Source Environment != Listener Environment, **then** calculate source's Exclusion and Occlusion levels into primary environment.*

Mark source as processed.

Step 1d: Update Listener

***If** listener environment changed, **then** update EAXCONTEXT_PRIMARYFXSLOTID.*

Step 1e: Update FX Slot Panning and Volume parameters

Update direction and magnitude of panning vectors (reverb and reflection) as well as volume (based on distance from listener to environment).

Step 2: Update sources

For each un-processed Source (processed sources were marked in step 1c):

*Determine source's environment. **If** changed:*

***If** an FX Slot is rendering this environment, **then** enable send (EAXSOURCE_ACTIVEFXSLOTID == FX Slot GUID of active environment, and EAX_PrimaryFXSlotID).*

***If** an FX Slot is not rendering this environment, **then** disable sends (EAXSOURCE_ACTIVEFXSLOTID = NULL, and EAX_PrimaryFXSlotID).*

***If** Source Environment == Listener Environment, **then** set Obstruction as required.*

***If** Source Environment != Listener Environment, **then** calculate source's Exclusion and Occlusion levels into primary environment as required.*

Reverb and Reflection Panning Algorithm

The EAX Reverb effect exposes complete control over the early reflections and the late reverberation, including parameters that control the spatial distribution of these separate parts.

The spatial distribution of the early reflections is controlled by using a single vector that indicates the direction of the reflections, while its magnitude controls how focused the reflections are toward this direction. A vector of magnitude 0 (the default) creates reflections that come evenly from all directions, whereas a vector of magnitude 1.0 is highly focused to a particular point. The spatial distribution of the late reverb is controlled in the same way, using a separate vector.

Both vectors are interpreted in the co-ordinate system of the user, without taking into account the orientation of the 3D listener. For example, setting the reverb pan to (0.0, 0.0, 0.7) means that the reverb is panned to the front speakers, whereas a vector of (0.0, 0.0, -0.7) pans the reverb to the rear speakers. Note, that the pan vectors use a left-handed co-ordinate system, so if an application is using Open AL it will need to consider this.

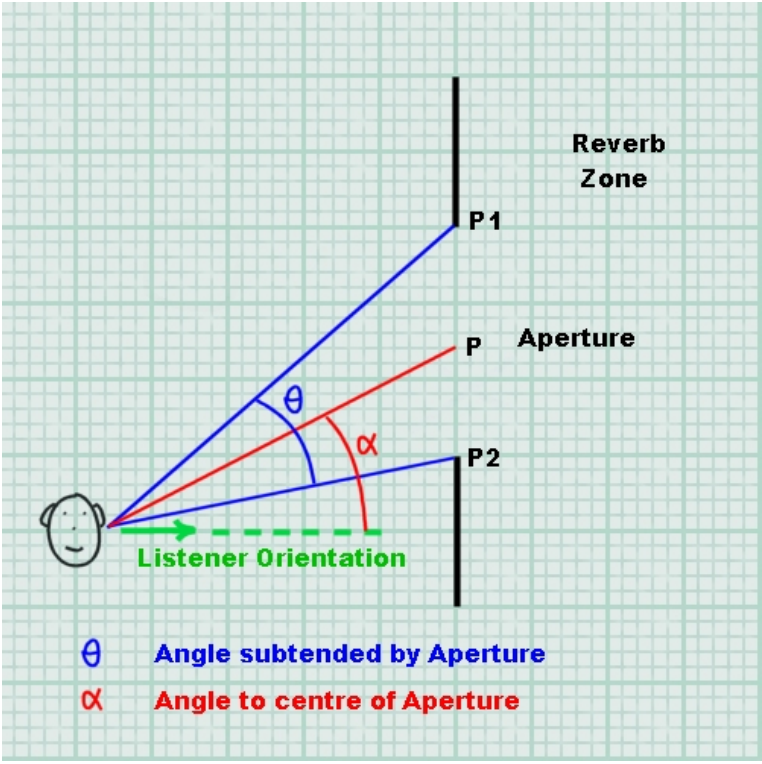

Figure 4 - Example scenario for effects panning

Assume the Listener position is stored in the vector vListenerPos, the orientation in vListenerOri, the centre of the aperture (P) in vAperture, and the aperture corners (P1 and P2) in vApertureLeftSide and vApertureRightSide.

All angles and trigonometric functions are using radians.

Orientation

In order to compute the orientation for the Reflections and Reverb pan vectors it is necessary to know the orientation of the listener, and the desired location of the environment. In Figure 3, the reverb on the other side of the aperture should be panned to the centre of the aperture, point P on the diagram.

To translate P to the "user-relative" position required by the Reflections and Reverb Pan vectors, the vector LP should be inversely rotated by the angle between the listener orientation and the straight-ahead vector (0,0,1).

To calculate this angle, take the dot product of the Listener Orientation (should already be normalized) and (0,0,1). Since several terms cancel out, the angle is simply: -

Angle = arccos (vListenerOri.z).

If the Listener is facing to the left of straight ahead then the angle should be negated: -

```
if (vListenerOri.x < 0)
    Angle = -Angle
```

The vector LP is defined as: -

LP.x = vAperture.x – vListenerPos.x

LP.y = vAperture.y – vListenerPos.y

LP.z = vAperture.z – vListenerPos.z

So, the EAX Vector can be calculated by rotating LP by –Angle: -

Pan.x = (LP.x * cos(-Angle)) + (LP.z * sin(-Angle))

Pan.y = 0

Pan.z = (LP.x * -sin(-Angle)) + (LP.z * cos(-Angle)

The resulting Pan vector is a "user-relative" vector pointing to the location where the reverb should be panned. This vector should be normalized, and then multiplied by the magnitude calculated in the next step.

Magnitude

The magnitude of the Reflection and Reverb pan vectors indicate how focused the reflected sound should be. To compute the magnitude, the angle subtended by the aperture (θ in the diagram) needs to be calculated.

The vectors LP1 and LP2 are defined as: -

LP1.x = vApertureLeftSide.x – vListenerPos.x

LP1.y = vApertureLeftSide.y – vListenerPos.y

LP1.z = vApertureLeftSide.z – vListenerPos.z

LP2.x = vApertureRightSide.x – vListenerPos.x

LP2.y = vApertureRightSide.y – vListenerPos.y

LP2.z = vApertureRightSide.z – vListenerPos.z

After LP1 and LP2 have been normalized, the angle is computed.

θ = arccos((LP1.x * LP2.x) + (LP1.y * LP2.y) + (LP1.z * LP2.z));

The magnitude of the EAX Vector can be calculated using the following formula: -

Magnitude = (2.0f * sin (θ / 2.0)) / θ

Assuming Pan is a normalized vector, the Reflections and Reverb vectors should be set to: -

Pan.x = Pan.x * Magnitude

Pan.y = Pan.y * Magnitude

Pan.z = Pan.z * Magnitude

Table 1 shows some example magnitudes for various angles.

Angle (θ) Degrees	Angle (θ) Radians	Magnitude
0	0	1.0
45	π / 4	0.97
90	π / 2	0.90
180	π	0.637
270	3 π / 2	0.3
360	2 π	0

Table 1 - magnitude values for different panning focus angles

In addition to controlling the position and focus of the early reflections and late reverb, the overall volume can be attenuated to indicate the distance between the listener and the acoustically reverberant environment.

Upgrading an EAX 3.0 application to EAX 4.0

If an application already supports EAX 3.0, then it is trivial to upgrade the application to support EAX 4.0. EAX 4.0 is a super-set of EAX 3.0 functionality, and both versions of EAX use the same environmental reverberation model. While some objects have been renamed in EAX 4.0, to be more consistent in a multi-FX slot context, each EAX 3.0 object has an analogous object in EAX 4.0. The EAX 3.0 Listener object was used to control the parameters of the single reverb effect, so this can be replaced with the EAX 4.0 FX Slot 0 object (which will contain the reverb effect for legacy reasons). The EAX 3.0 Buffer object was used to control Source specific EAX effects, so this can be replaced with the EAX 4.0 Source object.

An existing EAX 3.0 application can be upgraded to EAX 4.0 by replacing the EAX 3.0 eax.h file with the new EAX 4.0 eax.h file, and adding an additional header file called eaxlegacy.h. The eaxlegacy.h file redefines all the EAX 3.0 objects, parameter names, flag names, and structure names to their EAX 4.0 counterparts.

If an application does not contain legacy EAX code, then it is recommended that the application ignore the eaxlegacy.h file and use the new EAX 4.0 terminology.

Backwards Compatibility

As with previous versions of EAX, Creative provides an automatic backwards compatibility technology known as EAX Unified. EAX Unified is available for Direct Sound and Open AL, and provides a graceful fallback from one version of EAX to a lesser version. For more details about EAX Unified, please refer to the separate documentation in the SDK.

Performance and Optimization

Over recent years, advanced 3D sound technologies like EAX and hardware sound buffers have gained a reputation for hogging the processor and causing performance problems. However, a close understanding of the potential bottlenecks in 3D sound programming will enable the developer to avoid inefficient use of hardware resources. EAX coding techniques are inextricably linked with hardware 3D sound management, so this section will deal with both of these topics together. With a few key optimizations, implementing hardware 3D sound with EAX support in your project will not burn unnecessary CPU time or cause excessive slowdown to the frame rate.

Hardware vs. Software audio

A standard software 3D audio renderer generally employs simple algorithms to simulate 3D spatialization using stereo panning and attenuation. On a PC system with a modern CPU, multiple audio streams can be panned, balanced, and mixed together with a fraction of the available computing resources. Software mixers typically update at a relatively slow rate, maybe 40 times per second, thus introducing up to 25 milliseconds of latency. Changes to audio, for instance setting a new listener position or orientation, or playing and stopping sound buffers, will only be effective to a resolution of 25ms.

Hardware 3D sound renderers, on the other hand, spatialize sounds over several speakers with sophisticated multi-speaker panning algorithms, or intensive HRTF processing on headphone and stereo playback systems. A typical hardware accelerator will respond immediately to changes in audio, so that all 3D sources will be instantly affected by any change in the listener's position or orientation. New sound sources will typically begin playing straight away. With EAX effects switched on, the hardware audio accelerator is additionally performing complex audio processing such as filtering and digital reverberation. Rendering up to 64 equivalent 3D voices with EAX effects in software would be prohibitively CPU intensive, even on today's fastest processors.

So when comparing audio performance between software and hardware 3D audio renderers, one must bear in mind the quality of the results. That said, much of the work in hardware 3D sound is carried out on the soundcard's own dedicated audio processor. The driver level processing which does take up CPU cycles is comparatively simple, and this is where careful programming techniques can improve performance.

Key performance bottlenecks for hardware audio and EAX

The key performance hits in 3D hardware audio occur when information such as 3D position data and EAX settings are sent from the application to the soundcard, via the operating system and the hardware's drivers, which run on the host CPU. The causes of delays are threefold.

- Firstly, the actual process of passing the data through the OS layers takes some time in itself.
- The soundcard responds immediately to changes in 3D Audio parameters, so the properties of the audio scene must be constantly re-calculated.
- Additionally, when a drastic change has been made to audio settings, the soundcard must perform interpolation techniques to prevent audio artifacts.

It now becomes obvious that any technique that avoids unnecessary updating of 3D audio settings will cut down CPU usage and enhance performance. An audio implementation that carries out optimal setting updates should perform just as well when rendered on a complex hardware accelerator or a very simple software mixer.

Solutions for optimization

Now the potential bottlenecks are well defined, it is possible to propose techniques to avoid them. The solutions involve cutting down the update frequency, and avoiding redundant updates.

Audio frame rate

Establishing an independent audio frame rate is a crucial performance technique that can potentially halve the CPU time spent on 3D audio updates. The human eye detects visual changes at a very high resolution, hence a high graphical frame rate is desirable – graphical updates should ideally occur at around 60 frames per second to give good smooth visuals.

Many developers update their 3D sound scene each time the screen is refreshed. However, human ears do not require such frequent updates to 3D sound positioning. In fact, a rate of 30 positional updates per second is more than adequate to preserve convincing 3D audio. Note that although sound *positioning* does not require updates over 30Hz, some properties such as sound playback frequency, popular for simulating vehicle engine speed, could need updates that are more frequent.

On the other hand, some EAX settings such as environmental reverb, which can be more expensive to modify, actually require an even lower update rate, as low as 10 to 15Hz. One exception to this is the Reflections and Reverb pan vectors, which if being used to localize secondary environments, should be updated as often as the Listener position and orientation.

Application property	Required update rate
Graphical scene	Typically 60 Hz+
Sound playback frequency for pitch adjustments on sound sources	60 Hz
3D listener Position / Orientation	30 Hz
3D sound source position / directivity	30 Hz
Reverb / reflections panning vectors	30 Hz
Reverberations parameter adjustments	15 Hz

Table 2 - Optimum update rates for different 3D audio properties

Therefore, developing techniques to limit the frequency of positional audio updates to each source and the listener to pre-determined rates will save plenty of CPU cycles. Table 2 shows optimum independent rates for updating resolution-critical settings. The next section will deal with some mechanisms that further aid the implementation of efficient 3D audio.

Deferred settings

3D audio APIs DirectSound3D and OpenAL both offer facilities for deferring settings, as does the EAX API.

In DirectSound3D and EAX API calls, a flag can be passed to each parameter update function to mark the update as deferred. When the developer wants the deferred changes to actually be carried out, a 'commit' call is made, and the sound scene is re-calculated.

With OpenAL, the developer can use the EAX defer flags, or OpenAL's Suspend and Process Context functionality. The Suspend and Process functionality includes all calls to EAX. If Suspend and Process is not used, then EAX calls are deferred and committed in the same way as Direct Sound.

See the appropriate API documentation for more details on these systems.

As noted above, most 3D Audio hardware accelerators will process audio calls immediately. This means that every time a call is made to update the position or orientation of the listener, the soundcard must perform a large number of calculations to determine for each playing sound source, the attenuation, the attenuation at high frequencies (air absorption), position, velocity, reverb send level, etc. These calculations must be recomputed every time one of the dependant variables is changed. Assume the following operations take place: -

Audio Function Call	Calculations Performed
Set Source 0 Position	Source 0 spatialization calculations
Set Source 0 Orientation	Source 0 spatialization calculations
Set Source 1 Position	Source 1 spatialization calculations
Set Source 1 Orientation	Source 1 spatialization calculations
Set Listener Position	All Source spatialization calculations
Set Listener Orientation	All Source spatialization calculations

Table 3 - Resulting calculations for setting different 3D audio properties

It is clear from the above sequence of calls, that the calculations to spatialize each Source are performed three times!

The above situation can be avoided by using 'deferred' settings. If the above operations were performed with the deferred flag set, the same table would look like this: -

Audio Function Call	Calculations Performed
Set Source 0 Position (deferred)	None. Source 0 position is stored.
Set Source 0 Orientation (deferred)	None. Source 0 orientation is stored.
Set Source 1 Position (deferred)	None. Source 1 position is stored.
Set Source 1 Orientation (deferred)	None. Source 1 orientation is stored.
Set Listener Position (deferred)	None. Listener position is stored.
Set Listener Orientation (deferred)	None. Listener orientation is stored.
Commit Deferred Settings	All Source spatialization calculations

Table 4 - Resulting calculations for setting different 3D audio properties as deferred

Utilizing deferred settings therefore provides a great optimization by preventing duplicate calculations being performed because not all the dependant variables are known.

Note that EAX deferred settings are committed by either a call to Commit that particular object, or, if / when a subsequent call on that object, or a global object is not deferred.

Specifically, EAX Source deferred settings are committed whenever any of the following types of calls are made: -

- An EAX Source, EAX Context, or EAX FX Slot Commit call
- A non-deferred EAX Source call
- Direct Sound: A non-deferred DS3D Buffer call (or any DS Buffer call, because they cannot be deferred).
- Direct Sound: A non-deferred DS3D Listener call
- Open AL: A 3D Source parameter call
- Open AL: A 3D Listener parameter call
- A non-deferred EAX FX Slot call (including Effect parameter calls)
- The Source begins to play.

It can be seen therefore that adjusting a global EAX object results in all deferred EAX Source settings being committed.

Similarly, EAX FX Slot deferred settings are committed whenever any of the following calls are made: -

- An EAX Context or EAX FX Slot Commit call
- A non-deferred EAX Context call
- A non-deferred EAX FX Slot call (including Effect parameter calls)

Set All Calls

If an application is adjusting multiple parameters of one EAX object, it can be more efficient to use the 'All parameters' call rather than making multiple calls to apply individual parameters, even if these calls are deferred.

Caching values

A good hardware accelerator will only perform 3D audio calculations if one or more dependant variables actually change value. However, due to the speed of passing data from an application to a driver, it is more efficient if the application can intercept calls that would otherwise be setting a parameter to the same value that it is already set to. A good audio engine should therefore cache all the 3D and EAX parameters (global and source specific), so it can check whether a value has actually changed before making the call to the Audio API.

Note: The current Windows implementations of Open AL include parameter-caching schemes to prevent redundant audio calls, so for an application using AL, there is no benefit to be gained by implementing the scheme described here.

Tables of EAX Properties

FX Slot Properties

Property Name	Type	Range	Default
EAXFXSLOT_ALLPARAMETERS	EAXFXSLOTPROPERTIES		
EAXFXSLOT_LOADEFFECT	GUID	N/A	
EAXFXSLOT_VOLUME	LONG	[-10000, 0]	0 mB
EAXFXSLOT_LOCK	LONG	[0, 1]	(*)
EAXFXSLOT_FLAGS	ULONG	[0x0, 0x1]	
• EAXFXSLOTFLAGS_ENVIRONMENT	Flag bit	0 or 1	TRUE
(**) Effect Parameter 1	Variable	Variable	Variable
(**) Effect Parameter 2	Variable	Variable	Variable
(**) Effect Parameter 3	Variable	Variable	Variable
…			

(*) The default setting of EAXFXSLOT_LOCK is TRUE for EAXPROPERTYID_EAX40_FXSlot0 and EAXPROPERTYID_EAX40_FXSlot1, FALSE for EAXPROPERTYID_EAX40_FXSlot2 and EAXPROPERTYID_EAX40_FXSlot3.

(**) After an effect type is loaded into the slot with EAXFXSLOT_LOADEFFECT or EAXFXSLOT_ALLPARAMETERS, all that effect's properties can be set for that slot. For example, if a reverb effect type is loaded (EAXREVERB_EFFECT), then all the reverb effect properties are now valid for that slot, e.g. EAXREVERB_NONE, EAXREVERB_ALLPARAMETERS, and EAXREVERB_ENVIRONMENT.

EAXFXSLOT_ALLPARAMETERS

Parameter Name	EAXFXSLOT_ALLPARAMETERS
Parameter Type	EAXFXSLOTPROPERTIES structure

The property EAXFXSLOT_ALLPARAMETERS allows you to set or get the values of all the FX Slot properties (this does not include the parameters of the effect loaded in the slot).

The EAXFXSLOTPROPERTIES structure is defined as follows: -

```
typedef struct _EAXFXSLOTPROPERTIES
{
  GUID          guidLoadEffect;
  long          lVolume;
  long          lLock;
  unsigned long ulFlags;
} EAXFXSLOTPROPERTIES, *LPEAXFXSLOTPROPERTIES;
```

EAXFXSLOT_LOADEFFECT

Parameter Name	EAXFXSLOT_LOADEFFECT
Parameter Type	GUID
Default Value	N/A

The property EAXFXSLOT_LOADEFFECT is used to the load the effect type specified by the GUID into the FX Slot. The following Effect GUIDs have been defined: -

GUID	**Effect Name**
EAX_AGCCOMPRESSOR_EFFECT	Automatic Gain Control Compressor
EAX_AUTOWAH_EFFECT	Autowah
EAX_CHORUS_EFFECT	Chorus
EAX_DISTORTION_EFFECT	Distortion
EAX_ECHO_EFFECT	Echo
EAX_EQUALIZER_EFFECT	Equalizer
EAX_FLANGER_EFFECT	Flanger
EAX_FREQUENCYSHIFTER_EFFECT	Frequency Shifter
EAX_VOCALMORPHER_EFFECT	Vocal Morpher
EAX_PITCHSHIFTER_EFFECT	Pitch Shifter
EAX_RINGMODULATOR_EFFECT	Ring Modulator
EAX_REVERB_EFFECT	Reverb

EAXFXSLOT_VOLUME

Parameter Name	EAXFXSLOT_VOLUME
Parameter Type	LONG
Default Value	0
Range	-10000 to 0
Units	mB (1/100th of a decibel)

The property EAXFXSLOT_VOLUME controls the output volume of the FX Slot.

EAXFXSLOT_LOCK

Parameter Name	EAXFXSLOT_LOCK
Parameter Type	LONG
Default Value	FX Slots 0 and 1 are EAXFXSLOT_LOCKED, FX Slots 2 and 3 are EAXFXSLOT_UNLOCKED
Range	[EAXFXSLOT_UNLOCKED, EAXFXSLOT_LOCKED]
Units	Enumeration value

The property EAXFXSLOT_LOCK is used to prevent other EAX 4.0 applications from changing the type of Effect loaded into this FX Slot. It does not prevent Effect parameters from being changed.

An EAX 4.0 application can always get the status of the EAXFXSLOT_LOCK property on any FX Slot.

An EAX 4.0 application can only set the EAXFXSLOT_LOCK property to EAXFXSLOT_LOCKED if the property is currently EAXFXSLOT_UNLOCKED, or if the property was previously set to EAXFXSLOT_LOCKED by the same application.

An EAX 4.0 application can only set the EAXFXSLOT_LOCK property to EAXFXSLOT_UNLOCKED if the property is currently EAXFXSLOT_UNLOCKED, or if the property was previously set to EAXFXSLOT_UNLOCKED by the same application.

Please note that, in the current implementation, FX Slot 0 and FX Slot 1 are locked as if another EAX 4.0 application set the EAXFXSLOT_LOCK property to EAXFXSLOT_LOCKED. This means the EAXFXSLOT_LOCK property cannot be changed and the effect types loaded will always be the default ones, i.e. Environmental Reverb and Chorus respectively. In future audio architectures, this limitation may be removed, so an application should not assume this behavior.

Illegal set calls on the EAXFXSLOT_LOCK property will generate an EAXERR_INVALID_OPERATION error.

EAXFXSLOT_FLAGS

Parameter Name	EAXFXSLOT_FLAGS
Parameter Type	ULONG
Default Value	EAXFXSLOTFLAGS_ENVIRONMENT
Units	Each bit is a Boolean flag

Currently only one FX Slot flag has been defined: EAXFXSLOTFLAGS_ENVIRONMENT, which indicates if the effect contained in the FX Slot should be treated as environmental or not.

The flag EAXFXSLOTFLAGS_ENVIRONMENT is set to TRUE by default on all slots, irrespective of the effect type loaded in them. If EAXFXSLOTFLAGS_ENVIRONMENT is FALSE, setting *EAXCONTEXT_PRIMARYFXSLOTID* to this slot has no effect.

Source Properties

Source property names are prefixed with EAXSOURCE_ and flags names are prefixed with EAXSOURCEFLAGS_

	Property Name	Type	Range	Default
	ALLPARAMETERS	EAXSOURCEPROPERTIES		
	OBSTRUCTIONPARAMETERS	EAXOBSTRUCTIONPROPERTIES		
	OCCLUSIONPARAMETERS	EAXOCCLUSIONPROPERTIES		
	EXCLUSIONPARAMETERS	EAXEXCLUSIONPROPERTIES		
	DIRECT	LONG	[–10000, 1000]	0 mB
	DIRECTHF	LONG	[–10000, 0]	0 mB
	ROOM	LONG	[–10000, 1000]	0 mB
	ROOMHF	LONG	[–10000, 0]	0 mB
3D	OBSTRUCTION	LONG	[–10000, 0]	0 mB
3D	OBSTRUCTIONLFRATIO	FLOAT	[0.0, 1.0]	0.0
3D	OCCLUSION	LONG	[–10000, 0]	0 mB
3D	OCCLUSIONLFRATIO	FLOAT	[0.0, 1.0]	0.25
3D	OCCLUSIONROOMRATIO	FLOAT	[0.0, 10.0]	1.5
3D	OCCLUSIONDIRECTRATIO	FLOAT	[0.0, 10.0]	1.0
3D	EXCLUSION	LONG	[–10000, 0]	0 mB
3D	EXCLUSIONLFRATIO	FLOAT	[0.0, 1.0]	1.0
3D	OUTSIDEVOLUMEHF	LONG	[–10000, 0]	0 mB
3D	DOPPLERFACTOR	FLOAT	[0.0, 10.0]	1.0
3D	ROLLOFFFACTOR	FLOAT	[0.0, 10.0]	0.0
3D	ROOMROLLOFFFACTOR	FLOAT	[0.0, 10.0]	0.0
3D	AIRABSORPTIONFACTOR	FLOAT	[0.0, 10.0]	0.0
3D	FLAGS	ULONG	[0x0, 0x7]	
	• DIRECTHFAUTO	Flag bit		TRUE
	• ROOMAUTO	Flag bit		TRUE
	• ROOMHFAUTO	Flag bit		TRUE
	SENDPARAMETERS	EAXSOURCESENDPROPERTIES		
	ALLSENDPARAMETERS	Array of EAXSOURCEALL SENDPROPERTIES		
	OCCLUSIONSENDPARAMETERS	Array of EAXSOURCE OCCLUSIONSENDPROPERTIES		
	EXCLUSIONSENDPARAMETERS	Array of EAXSOURCE EXCLUSIONSENDPROPERTIES		
	ACTIVEFXSLOTID	Array of GUIDs		(*)

(*) Default values are {EAX_NULL_GUID, EAX_PrimaryFXSlotID}

(3D) Indicates parameter is disabled if the buffer's 3D mode is disabled (Direct Sound only).

EAXSOURCE_ALLPARAMETERS

Parameter Name	EAXSOURCE_ALLPARAMETERS
Parameter Type	EAXSOURCEPROPERTIES structure

The property EAXSOURCE_ALLPARAMETERS allows the application to set or get the values of all source properties at once (with exception of *EAXSOURCE_SENDPARAMETERS*, *EAXSOURCE_ALLSENDPARAMETERS*, *EAXSOURCE_OCCLUSIONSENDPARAMETERS*, *EAXSOURCE_EXCLUSIONSENDPARAMETERS* and *EAXSOURCE_ACTIVEFXSLOTID*).

The EAXSOURCEPROPERTIES structure is defined as: -

```
typedef struct _EAXSOURCEPROPERTIES
{
  long          lDirect;
  long          lDirectHF;
  long          lRoom;
  long          lRoomHF;
  long          lObstruction;
  float         flObstructionLFRatio;
  long          lOcclusion;
  float         flOcclusionLFRatio;
  float         flOcclusionRoomRatio;
  float         flOcclusionDirectRatio;
  long          lExclusion;
  float         flExclusionLFRatio;
  long          lOutsideVolumeHF;
  float         flDopplerFactor;
  float         flRolloffFactor;
  float         flRoomRolloffFactor;
  float         flAirAbsorptionFactor;
  unsigned long ulFlags;
} EAXSOURCEPROPERTIES, *LPEAXSOURCEPROPERTIES;
```

EAXSOURCE_OBSTRUCTIONPARAMETERS

Parameter Name	EAXSOURCE_OBSTRUCTIONPARAMETERS
Parameter Type	EAXOBSTRUCTIONPROPERTIES structure

The EAXSOURCE_OBSTRUCTIONPARAMETERS property is used to set all of the Obstruction parameters in one call. The EAXOBSTRUCTIONPROPERTIES structure is defined as: -

```
typedef struct _EAXOBSTRUCTIONPROPERTIES
{
    long          lObstruction;
    float         flObstructionLFRatio;
} EAXOBSTRUCTIONPROPERTIES, *LPEAXOBSTRUCTIONPROPERTIES;
```

EAXSOURCE_OCCLUSIONPARAMETERS

Parameter Name	EAXSOURCE_OCCLUSIONPARAMETERS
Parameter Type	EAXOCCLUSIONPROPERTIES structure

The EAXSOURCE_OCCLUSIONPARAMETERS property is used to set all of the Occlusion parameters in one call. The EAXOCCLUSIONPROPERTIES structure is defined as: -

```
typedef struct _EAXOCCLUSIONPROPERTIES
{
  long          lOcclusion;
  float         flOcclusionLFRatio;
  float         flOcclusionRoomRatio;
  float         flOcclusionDirectRatio;
} EAXOCCLUSIONPROPERTIES, *LPEAXOCCLUSIONPROPERTIES;
```

EAXSOURCE_EXCLUSIONPARAMETERS

Parameter Name	EAXSOURCE_EXCLUSIONPARAMETERS
Parameter Type	EAXEXCLUSIONPROPERTIES structure

The EAXSOURCE_EXCLUSIONPARAMETERS property is used to set all of the Exclusion parameters in one call. The EAXEXCLUSIONPROPERTIES structure is defined as: -

```
typedef struct _EAXEXCLUSIONPROPERTIES
{
  long          lExclusion;
  float         flExclusionLFRatio;
} EAXEXCLUSIONPROPERTIES, *LPEAXEXCLUSIONPROPERTIES;
```

EAXSOURCE_DIRECT

Parameter Name	EAXSOURCE_DIRECT
Parameter Type	LONG
Default Value	0
Range	-10000 to 1000
Units	mB (1/100th of a decibel)

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

If you specify an increase in direct-path sound which, when combined with the direct-path amount set by both OpenAL or DirectSound and EAX source properties, results in a total gain larger than the intensity of the unattenuated sound source (referred to as 0 dB), the EAX driver clips the resulting gain to 0 dB. This means that the Direct property can never increase the direct-path sound to more than the original sound source.

EAXSOURCE_DIRECTHF

Parameter Name	EAXSOURCE_DIRECTHF
Parameter Type	LONG
Default Value	0
Range	-10000 to 0
Units	mB (1/100th of a decibel)

The Direct HF property is a low-level property that applies a relative correction to the sound source's direct-path intensity at high frequencies. Like the Direct property, this correction comes in addition to the effects of the other OpenAL or DirectSound and EAX properties.

EAXSOURCE_ROOM

Parameter Name	EAXSOURCE_ROOM
Parameter Type	LONG
Default Value	0
Range	-10000 to 1000
Units	mB (1/100th of a decibel)

The Room property is a low-level sound-source property that adjusts the effect send levels into all environmental effect slots. It is an additive property; its setting is added to the total effect level for this source that is specified by all other EAX listener and source properties. You can use the Room sound-source property to correct the intensity of reflected sound at minimum distance as defined by the Room property of the EAX Reverb effect. This can be useful especially if different sound sources have different minimum distances. Note that although you can specify a positive Room value, the combined effect of all OpenAL (or DirectSound) and EAX source properties will be limited by the driver to a maximum gain of 0 dB. You can use positive values of the Room property to restore the amount of reflected sound that has been diminished by the effects of other source properties, but you can never amplify the original source volume.

EAXSOURCE_ROOMHF

Parameter Name	EAXSOURCE_ROOMHF
Parameter Type	LONG
Default Value	0
Range	-10000 to 0
Units	mB (1/100th of a decibel)

The Room HF property is a low-level sound-source property that adjusts the high-frequency effect send levels into all environmental effect slots. The value of Room HF ranges from 0 dB (no filter) to -100 dB (virtually no reflected sound). This is an additive property; its setting is added to the effects of the other sound-source properties to determine, for this source, the relative attenuation of environmental effects at the reference high frequency.

EAXSOURCE_OBSTRUCTION

Parameter Name	EAXSOURCE_OBSTRUCTION
Parameter Type	LONG
Default Value	0
Range	-10000 to 0
Units	mB (1/100th of a decibel)

The Obstruction property specifies the amount of obstruction muffling to apply to a sound source's direct-path sound. Obstruction occurs when an object lies between a sound source and a listener. The direct path from source to listener is muffled by the obstruction, but the reflected sound from the source remains unchanged. The value set for the Obstruction property determines the direct-path attenuation at the reference high frequency.

Obstruction can be generally used as an alternative way of specifying a filter applied to the direct path. The Direct and Direct HF properties respectively provide a frequency-independent attenuation and an additional attenuation of the high frequencies. On the other hand, Obstruction provides high-frequency attenuation and relative attenuation of low frequencies.

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

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

Note that you can use Obstruction and Occlusion or Exclusion simultaneously - if, for example, the source is in another room from the listener and there is a large obstacle between the listener and the wall. In this case, the dry path is filtered twice: once by Occlusion and once by Obstruction. Obstruction and Exclusion may be combined if the direct path must bend around the edge of an opening between two rooms before reaching the listener.

EAXSOURCE_OBSTRUCTIONLFRATIO

Parameter Name	EAXSOURCE_OBSTRUCTIONLFRATIO
Parameter Type	FLOAT
Default Value	0.0
Range	0.0 to 1.0
Units	A linear multiplier value

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

EAXSOURCE_OCCLUSION

Parameter Name	EAXSOURCE_OCCLUSION
Parameter Type	LONG
Default Value	0
Range	-10000 to 0
Units	mB (1/100th of a decibel)

The Occlusion property specifies the amount of occlusion muffling to apply to a sound source's direct sound path and to the reflected sound that it generates in the listener's environment (referenced by *EAXCONTEXT_PRIMARYFXSLOTID*). Occlusion occurs when the listener is in one room or environment, the sound source is in another room or environment, and the listener hears the sound through a separating wall or through a closed door or window. The value set for the Occlusion property determines the attenuation at the reference high frequency.

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

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

Occlusion's maximum value, 0, specifies no attenuation and hence no occlusion effect. The minimum value, –10000 (which is -100 dB), indicates that the sound source is so occluded that it is barely audible. Any value between minimum and maximum indicates partial occlusion.

EAXSOURCE_OCCLUSIONLFRATIO

Parameter Name	EAXSOURCE_OCCLUSIONLFRATIO
Parameter Type	FLOAT
Default Value	0.25
Range	0.0 to 1.0
Units	A linear multiplier value

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

EAXSOURCE_OCCLUSIONROOMRATIO

Parameter Name	EAXSOURCE_OCCLUSIONROOMRATIO
Parameter Type	FLOAT
Default Value	1.5
Range	0.0 to 10.0
Units	A linear multiplier value

The Occlusion Room Ratio and Occlusion Direct Ratio properties control the amount of attenuation and filtering applied to the reflected sound and the direct path, respectively, for a given setting of the Occlusion property. The attenuation obtained at the reference high frequency is determined by multiplying the Occlusion property value by the Occlusion Room Ratio or the Occlusion Direct Ratio.

When both values are set to 0.0, the Occlusion property has no effect. If Occlusion Room Ratio is set to 0.0 and Occlusion Direct Ratio to 1.0, Occlusion is equivalent to Obstruction. If Occlusion Room Ratio is set to 1.0 and Occlusion Direct Ratio to 0.0, Occlusion is equivalent to Exclusion (described in the next section).

When the Ratio value is set between 0.0 and 1.0, the effect is equivalent, for low and high frequencies, to scaling the setting of the Occlusion property by that value. If the value is larger than 1.0, that is only true at high frequencies; at low frequencies, the attenuation is such that the filter slope (difference between low frequencies and high frequencies) remains the same as for a setting of 1.0.

The default settings: 1.0 for Occlusion Direct Ratio and 1.5 for Occlusion Room Ratio—specify that, compared to the direct sound path, the reflected sound undergoes an additional frequency-independent attenuation that is equal to half the setting of Occlusion. This creates a natural sensation of occlusion because, in the physical world, it is the occluding wall that acts as the actual sound source in the listener's room. Since the wall radiates sound in only half the space that a sound source in the middle of the room can, it generates significantly less reflected sound than the original source would if it were located in the room.

EAXSOURCE_OCCLUSIONDIRECTRATIO

Parameter Name	EAXSOURCE_OCCLUSIONDIRECTRATIO
Parameter Type	FLOAT
Default Value	1.0
Range	0.0 to 10.0
Units	A linear multiplier value

This property is described in the EAXSOURCE_OCCLUSIONROOMRATIO section.

EAXSOURCE_EXCLUSION

Parameter Name	EAXSOURCE_EXCLUSION
Parameter Type	LONG
Default Value	0
Range	-10000 to 0
Units	mB (1/100th of a decibel)

The Exclusion and Exclusion LF Ratio properties are defined exactly as the corresponding Obstruction properties described above, except that they apply an attenuation and filtering effect to the reflected sound (of the primary environment) instead of the direct-path sound. The value set for the Exclusion property determines the attenuation applied to the reflected sound at the reference high frequency, while the direct-path sound is left unaffected. You can set the value of the Exclusion property for each sound source.

Exclusion's maximum value, 0, specifies no attenuation of the reflected sound. The minimum value, –10000 (which is -100 dB), indicates that the reflected sound is barely audible. Any value between minimum and maximum indicates partial exclusion.

The Exclusion LF Ratio property specifies the attenuation at low frequencies relative to the attenuation at high frequencies. The minimum value of 0.0 (the default value) specifies no attenuation at low frequencies; the maximum value of 1.0 specifies a frequency-independent attenuation. Note that adjusting Exclusion LF Ratio alone has no effect if Exclusion is set to 0.

The Exclusion property can be used when the sound source is not in the same room as the listener but its direct path can reach the listener through an opening (window, door). In such a situation, the sound source can only contribute a limited amount of energy to the reverberation of the listener's room (especially if it is distant from the aperture). Therefore, a low setting of the Exclusion property would be used. As the source comes closer to entering the room, the Exclusion value can be increased to apply more reverberation to this source. Note that you can use Obstruction simultaneously with Exclusion if the source is not directly visible through the aperture. In this way, some muffling can be applied to the direct path to reproduce the effect of diffraction around the edge of the aperture.

Exclusion can also be generally used as an alternative way of specifying a filter applied to the reflected sound (early reflections and late reverberation). The Room and Room HF properties respectively provide a frequency-independent attenuation and an additional attenuation of the high frequencies. On the other hand, Exclusion provides high-frequency attenuation and relative attenuation of the low frequencies.

EAXSOURCE_EXCLUSIONLFRATIO

Parameter Name	EAXSOURCE_EXCLUSIONLFRATIO
Parameter Type	FLOAT
Default Value	1.0
Range	0.0 to 1.0
Units	A linear multiplier value

This property is described in the EAXSOURCE_EXCLUSION section.

EAXSOURCE_OUTSIDEVOLUMEHF

Parameter Name	EAXSOURCE_OUTSIDEVOLUMEHF
Parameter Type	LONG
Default Value	0
Range	-10000 to 0
Units	mB (1/100th of a decibel)

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

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

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

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

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

EAXSOURCE_DOPPLERFACTOR

Parameter Name	EAXSOURCE_DOPPLERFACTOR
Parameter Type	FLOAT
Default Value	1.0
Range	0.0 to 10.0
Units	A linear multiplier value

The Doppler Factor property is a low-level sound-source property that is defined the same way as the Doppler Factor property provided in OpenAL or DirectSound. A value of 1.0 provides natural Doppler effects according to the movement of the source relative to the listener. A value larger than 1.0 will exaggerate these effects. A value of 0.0 will eliminate Doppler effects.

When using this per-source Doppler Factor, it is preferable to set the OpenAL or DirectSound global Doppler Factor to 1.0. Otherwise, the EAX Doppler factor is multiplied to the setting of the OpenAL or DirectSound Doppler Factor to get the final Doppler multiplier value for each source.

EAXSOURCE_ROLLOFFFACTOR

Parameter Name	EAXSOURCE_ROLLOFFFACTOR
Parameter Type	FLOAT
Default Value	0.0
Range	0.0 to 10.0
Units	A linear multiplier value

The Rolloff Factor property is a low-level sound-source property that is defined the same way as the listener Rolloff Factor property provided in DirectSound, and is equivalent to the per-source Rolloff Factor already provided in OpenAL. The EAX Source Rolloff factor is added to the setting of the OpenAL or DirectSound Rolloff Factor to get the final rolloff multiplier value for each source. When using the EAX Source Rolloff Factor in DirectSound, it is therefore preferable to set DirectSound's global Rolloff Factor to 0.0. A value of 1.0 specifies that the reflected sound will decay by 6 dB every time the distance doubles. Any value other than 1.0 is equivalent to a scaling factor that is applied to the quantity ((source listener distance) - (Minimum Distance)).

EAXSOURCE_ROOMROLLOFFFACTOR

Parameter Name	EAXSOURCE_ROOMROLLOFFFACTOR
Parameter Type	FLOAT
Default Value	0.0
Range	0.0 to 10.0
Units	A linear multiplier value

The Room Roll-off Factor property is a low-level sound-source property. It is one of two methods available in EAX to attenuate the reflected sound (early reflections and reverberation) according to

source-listener distance. Room Roll-Off is an additive property; its setting is added to the EAX Reverb Room Roll-Off setting to get the final room Roll-Off multiplier value for that source.

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

EAXSOURCE_AIRABSORPTIONFACTOR

Parameter Name	EAXSOURCE_AIRABSORPTIONFACTOR
Parameter Type	FLOAT
Default Value	0.0
Range	0.0 to 10.0
Units	A linear multiplier value

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

The air absorption value controls the distance-dependent attenuation at high frequencies caused by the propagation medium. It applies to both the direct-path and environmental effect paths, and can simulate sound transmission through foggy air, dry air, smoky atmosphere, and so on. The Air Absorption Factor default value of 0.0 turns off air absorption for this source, and a maximum value of 10.0 multiplies absorption by 10 for this source.

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

EAXSOURCE_FLAGS

Parameter Name	EAXSOURCE_FLAGS
Parameter Type	ULONG
Default Value	EAXSOURCEFLAGS_DIRECTHFAUTO | EAXSOURCEFLAGS_ROOMAUTO | EAXSOURCEFLAGS_ROOMHFAUTO
Units	Each bit is a Boolean flag

The Flags property uses its three low-order bits to set three sound-source-property flags. These flags determine whether you want the EAX engine to automatically adjust certain parameters of the direct path or environmental effect paths for a source. They are set to TRUE by default to provide a more realistic experience without any programming work.

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

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

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

EAXSOURCEFLAGS_ROOMAUTO
If this flag is TRUE (its default value), the intensity of this sound source's reflected sound (environmental effect paths) is automatically attenuated according to source-listener distance and source directivity (as determined by the cone parameters). If it is FALSE, the reflected sound is not attenuated according to distance and directivity.

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

If this flag is FALSE, the sound source's reflected sound is not filtered at all according to the source's directivity. Note that this is not the same as setting Outside Volume to 0, because this flag does not affect high-frequency attenuation of the source's direct-path sound in response to directivity attenuation. The Direct HF Auto flag, described earlier, controls that.

EAXSOURCE_SENDPARAMETERS

Parameter Name	EAXSOURCE_SENDPARAMETERS
Parameter Type	EAXSOURCESENDPROPERTIES structure(s)

The property EAXSOURCE_SENDPARAMETERS is used to adjust the amount of source signal that feeds a specific FX Slot (specified by the value of guidReceivingSlotID).

The EAXSOURCESENDPROPERTIES structure is defined as: -

```
typedef struct _EAXSOURCESENDPROPERTIES
{
    GUID          guidReceivingFXSlotID;
    long          lSend;
    long          lSendHF;
} EAXSOURCESENDPROPERTIES, *LPEAXSOURCESENDPROPERTIES;
```

The property EAXSOURCE_SENDPARAMETERS can set the send values for several FX Slots simultaneously by taking a variable number (determined by the size parameter of the 'Set' call) of the EAXSOURCESENDPROPERTIES structure.

The value lSend controls the amount of effect irrespective of frequency, whereas lSendHF provides an additional attenuation at high frequencies. The default value for lSend is 0 mB for all FX Slots. The default value for lSendHF is also 0 mB, which means that no low-pass filtering is applied.

EAXSOURCE_ALLSENDPARAMETERS

Parameter Name	EAXSOURCE_ALLSENDPARAMETERS
Parameter Type	EAXSOURCEALLSENDPROPERTIES structure

The property EAXSOURCE_ALLSENDPARAMETERS can accept multiple EAXSOURCEALLSENDPROPERTIES structures. The EAXSOURCEALLSENDPROPERTIES structure is defined as: -

```
typedef struct _EAXSOURCEALLSENDPROPERTIES
{
      GUID          guidReceivingFXSlotID;
      long          lSend;
      long          lSendHF;
      long          lOcclusion;
      float         flOcclusionLFRatio;
      float         flOcclusionRoomRatio;
      float         flOcclusionDirectRatio;
      long          lExclusion;
      float         flExclusionLFRatio;
} EAXSOURCEALLSENDPROPERTIES, *LPEAXSOURCEALLSENDPROPERTIES;
```

EAXSOURCE_OCCLUSIONSENDPARAMETERS

Parameter Name	EAXSOURCE_OCCLUSIONSENDPARAMETERS
Parameter Type	EAXSOURCEOCCLUSIONSENDPROPERTIES structure

The property EAXSOURCE_OCCLUSIONSENDPARAMETERS controls the gains with which a sound source feeds an FX Slot with occlusion. Unlike the EAX3.0 occlusion legacy properties (*EAXSOURCE_OCCLUSION*, etc…), which can only apply to the FX Slot corresponding to the listener's environment, EAXSOURCE_OCCLUSIONSENDPARAMETERS may apply to any environmental FX Slot. When sending to the FX Slot referenced in the Context PrimaryFXSlotID parameter, it combines additively with the occlusion legacy properties. It takes a variable number (determined by the size parameter of the 'Set' call) of the following structure: -

```
typedef struct _EAXSOURCEOCCLUSIONSENDPROPERTIES
{
  GUID guidReceivingFXSlotID;
  long lOcclusion;
  float     flOcclusionLFRatio;
  float     flOcclusionRoomRatio;
  float     flOcclusionDirectRatio;
} EAXSOURCEOCCLUSIONSENDPROPERTIES,
*LPEAXSOURCEOCCLUSIONSENDPROPERTIES;
```

NOTE: the same effect can be achieved by using the EAXSOURCE_DIRECT, EAXSOURCE_DIRECTHF, and EAXSOURCE_SENDPARAMETERS properties. However, we

introduce this new property to allow developers to keep using their same material presets defined with EAX3.0 occlusion properties and apply them to FX Slots different from the listener's environment.

EAXSOURCE_EXCLUSIONSENDPARAMETERS

Parameter Name	EAXSOURCE_EXCLUSIONSENDPARAMETERS
Parameter Type	EAXSOURCEEXCLUSIONSENDPROPERTIES structure

The property EAXSOURCE_EXCLUSIONSENDPARAMETERS controls the gains with which a sound source feeds an FX Slot with exclusion effect.

Unlike the EAX3.0 exclusion legacy properties (*EAXSOURCE_EXCLUSION*, etc…), which can only apply to the FX Slot corresponding to the listener's environment, EAXSOURCE_EXCLUSIONSENDPARAMETERS may apply to any environmental FX Slot. When sending to the FX Slot referenced in the Context PrimaryFXSlotID parameter, it combines additively with the exclusion legacy properties. It takes a variable number (determined by the size parameter of the 'Set' call) of the following structure: -

```
typedef struct _EAXSOURCEEXCLUSIONSENDPROPERTIES
{
  GUID guidReceivingFXSlotID;
  long lExclusion;
  float flExclusionLFRatio;
} EAXSOURCEEXCLUSIONSENDPROPERTIES,
*LPEAXSOURCEEXCLUSIONSENDPROPERTIES;
```

NOTE: the same effect can be achieved by using the Send and SendHF properties. However, we introduce this new property to allow developers to keep using their same material presets defined with EAX3.0 exclusion properties and apply them to FX Slots different from the listener's environment.

EAXSOURCE_ACTIVEFXSLOTID

Parameter Name	EAXSOURCE_ACTIVEFXSLOTID
Parameter Type	GUID or EAXACTIVEFXSLOTS structure
Default Value	{EAX_NULL_GUID, EAX_PrimaryFXSlotID}

The property EAXSOURCE_ACTIVEFXSLOTID specifies the active FX Slots for the source. The current specification allows a source to feed two FX Slots. The eax.h file therefore defines EAX_MAX_ACTIVE_FXSLOTS as 2. EAXSOURCE_ACTIVEFXSLOTID can take one or two GUIDs. The EAXACTIVEFXSLOTS structure is simply an array of GUIDs: -

```
typedef struct _EAXACTIVEFXSLOTS
{
  GUID guidActiveFXSlots[EAX_MAX_ACTIVE_FXSLOTS];
} EAXACTIVEFXSLOTS, *LPEAXACTIVEFXSLOTS;
```

It is not necessary to always pass in both GUIDs; passing in one GUID simply leaves the second Active FX Slot GUID unchanged. In other words, the EAX Engine keeps a 2-element array of the Active FX Slots for each source, so if the application passes in one GUID it will replace the first element of the array. It is the application wishes to change the 2nd Active FX Slot, then it must pass in two GUIDs.

To facilitate multi-environment games, a special GUID has been defined indicating that the source should send to the FX Slot indicated by the Context's PrimaryFXSlotID. This GUID is known as EAX_PrimaryFXSlotID (and is the default for the second Active GUID). If the Context's PrimaryFXSlotID is updated by the application (in response to the Listener's changing environment for example), then all Sources that have an Active GUID set to EAX_PrimaryFXSlotID, will automatically begin feeding the new FX Slot (if they weren't already).

If a multi-environment application wishes to enable the sends from each Source to the Listener' environment and the environment surrounding the source, the second GUID can be ignored (it defaults to EAX_PrimaryFXSlotID) and the first GUID updated in real-time to indicate the FX Slot rendering the source's environment.

NOTE: Setting EAXSOURCE_ACTIVEFXSLOTID to {EAX_NULL_GUID, EAX_NULL_GUID} mutes all FX Slot sends from the source.

NOTE: Setting EAXSOURCE_ACTIVEFXSLOTID with GUIDs that refer to the same FX Slot is the same as replacing one of them by EAX_NULL_GUID.

Context Properties

Context property names are prefixed with EAXCONTEXT_

Properties	Type	Range	Default
ALLPARAMETERS	EAXCONTEXTPROPERTIES		
PRIMARYFXSLOTID	GUID		EAXPROPERTYID_ EAX40_FXSlot0
DISTANCEFACTOR	FLOAT	[FLT_MIN, FLT_MAX]	1.0
AIRABSORPTIONHF	FLOAT	[-100.0, 0.0]	–5.0 mB/m
HFREFERENCE	FLOAT	[1000.0, 20000.0]	5000.0 Hz
LASTERROR	LONG		EAX_OK

EAXCONTEXT_ALLPARAMETERS

Parameter Name	EAXCONTEXT_ALLPARAMETERS
Parameter Type	EAXCONTEXTPROPERTIES structure

The property EAXCONTEXT_ALLPARAMETERS allows the application to set or get the values of all Context properties at once.

The EAXCONTEXTPROPERTIES structure is defined as: -

```
typedef struct _EAXCONTEXTPROPERTIES
{
  GUID          guidPrimaryFXSlotID;
  float         flDistanceFactor;
  float         flAirAbsorptionHF;
  float         flHFReference;
} EAXCONTEXTPROPERTIES, *LPEAXCONTEXTPROPERTIES;
```

EAXCONTEXT_PRIMARYFXSLOTID

Parameter Name	EAXCONTEXT_PRIMARYFXSLOTID
Parameter Type	GUID
Default Value	EAXPROPERTYID_EAX40_FXSlot0

The EAXCONTEXT_PRIMARYFXSLOTID property indicates which FX Slot is rendering the listener's environment. If none of the FX Slots are rendering the listener's environment, then this parameter should be set to the NULL GUID (EAX_NULL_GUID).

EAXCONTEXT_DISTANCEFACTOR

Parameter Name	EAXCONTEXT_DISTANCEFACTOR
Parameter Type	FLOAT
Default Value	1.0
Range	FLT_MIN to FLT_MAX
Units	A linear multiplier value

The EAXCONTEXT_DISTANCEFACTOR property is equivalent to the Direct Sound Distance Factor and combines multiplicatively with it. This property does not exist in OpenAL, so it is introduced here to allow OpenAL games to set a Distance Factor.

The Context Distance Factor is the number of meters in a vector unit. By default, the Context Distance Factor is 1.0. If a source is positioned at (2.0, 0.0, 0.0), then it is considered to be 2 meters to the right of the default Listener position. Applications that are using a different unit of measurement for 3-D graphics vectors should change the Distance Factor accordingly. This is necessary for some of the distance-related effects in EAX, such as air absorption. See *Distance Units* section.

EAXCONTEXT_AIRABSORPTIONHF

Parameter Name	EAXCONTEXT_AIRABSORPTIONHF
Parameter Type	FLOAT
Default Value	-5.0
Range	-100.0 to 0.0
Units	mB / m

The EAXCONTEXT_AIRABSORPTIONHF property controls the distance-dependent attenuation at the reference high frequency caused by the propagation medium (while causing no attenuation at low frequencies). It applies to both the direct path and the effect paths.

You can use EAXCONTEXT_AIRABSORPTIONHF to simulate sound transmission through foggy air, dry air, smoky atmosphere, and so on.

The EAXCONTEXT_AIRABSORPTIONHF and *EAXCONTEXT_HFREFERENCE* properties are in EAX 4.0 to make it possible to render the effect of air absorption even when the listener is not in an EAX environment. This can happen when the *EAXCONTEXT_PRIMARYFXSLOTID* is NULL (perhaps because the listener is in open air), or because the EAXFXSLOTFLAGS_ENVIRONMENT flag for the FX Slot identified by *EAXCONTEXT_PRIMARYFXSLOTID* is not set, or because that FX Slot does not contain a Reverb effect.

If the listener is not in a reverb environment, the properties EAXCONTEXT_AIRABSORPTIONHF and *EAXCONTEXT_HFREFERENCE* are applied to all sources; otherwise, they are ignored and the *EAXREVERB_AIRABSORPTIONHF* and *EAXREVERB_HFREFERENCE* values for the current listener environment are used instead.

The default value of EAXCONTEXT_AIRABSORPTIONHF is -0.05 dB per meter, which roughly corresponds to typical conditions of atmospheric humidity, temperature, and so on. Lowering the

value simulates a more absorbent medium (more humidity in the air, for example). Raising the value simulates a less absorbent medium (dry desert air, for example).

For Air Absorption to be used correctly, it is important to ensure that the 3D Audio Distance units are set to meters. See *Distance Units* section.

EAXCONTEXT_HFREFERENCE

Parameter Name	EAXCONTEXT_HFREFERENCE
Parameter Type	FLOAT
Default Value	5000.0
Range	1000.0 to 20000.0
Units	Hertz

The EAXCONTEXT_HFREFERENCE property determines the frequency at which the high-frequency effects created by EAX properties are measured. It only operates when the listener is not in a reverb environment, as described in the *EAXCONTEXT_AIRABSORPTIONHF* section. If the listener is in a reverb environment, the value of *EAXREVERB_HFREFERENCE* for that environment determines the reference high frequency for all EAX properties.

The EAXCONTEXT_HFREFERENCE value applies to the source properties for all sources. As a result, it is recommended to adopt the same setting of EAXCONTEXT_HFREFERENCE and *EAXREVERB_HFREFERENCE* for all environment presets used in a given application. Otherwise, changing the Environment preset while the application is running will affect certain properties that should remain characteristics of the sources (such as their directivity at high frequencies, specified by the *EAXSOURCE_OUTSIDEVOLUMEHF* property).

EAXCONTEXT_LASTERROR

Parameter Name	EAXCONTEXT_LASTERROR
Parameter Type	LONG
Default Value	EAX_OK
Units	Error Code Enumeration value

The error codes returned by Direct Sound and OpenAL for EAX function calls typically indicate whether the call succeeded or failed. The EAXCONTEXT_LASTERROR property is introduced in EAX 4.0 to provide the developer with more detailed information about why an EAX function call failed.

This property cannot be set, and after it has been retrieved, the error code clears (reset to EAX_OK).

For more information about the EAX Error Codes, please refer to the *Error Checking* section.

Reverb Effect Properties

Reverb property names are prefixed with EAXREVERB_ and flag property names are prefixed with EAXREVERBFLAGS_

Properties	Type	Range	Default
ALLPARAMETERS	EAXREVERBPROPERTIES		
ENVIRONMENT	DWORD	[0, EAX_ ENVIRONMENT_ COUNT-1]	EAX_ ENVIRONMENT_ GENERIC
ENVIRONMENTSIZE	FLOAT	[1.0, 100.0]	7.5 meters
ENVIRONMENTDIFFUSION	FLOAT	[0.0, 1.0]	1.0
ROOM	LONG	[–10000, 0]	–1000 mB
ROOMHF	LONG	[–10000, 0]	–100 mB
ROOMLF	LONG	[–10000, 0]	0 mB
DECAYTIME	FLOAT	[0.1, 20.0]	1.49 secs
DECAYHFRATIO	FLOAT	[0.1, 2.0]	0.83
DECAYLFRATIO	FLOAT	[0.1, 2.0]	1.00
REFLECTIONS	LONG	[–10000, 1000]	–2602 mB
REFLECTIONSDELAY	FLOAT	[0.0, 0.3]	0.007 secs
REFLECTIONSPAN	EAXVECTOR	(length 0. to 1.)	(0.0,0.0,0.0)
REVERB	LONG	[–10000, 2000]	200 mB
REVERBDELAY	FLOAT	[0.0, 0.1]	0.011 secs
REVERBPAN	EAXVECTOR	(length 0. to 1.)	(0.0,0.0,0.0)
ECHOTIME	FLOAT	[0.075, 0.25]	0.25 s
ECHODEPTH	FLOAT	[0.0, 1.0]	0.0
MODULATIONTIME	FLOAT	[0.04, 4.0]	0.25 s
MODULATIONDEPTH	FLOAT	[0.0, 1.0]	0.0
AIRABSORPTIONHF	FLOAT	[–100.0, 0.0]	–5.0 mB/m
HFREFERENCE	FLOAT	[1000.0, 20000.0]	5000.0 Hz
LFREFERENCE	FLOAT	[20.0, 1000.0]	250.0 Hz
ROOMROLLOFFFACTOR	FLOAT	[0.0, 10.0]	0.0
FLAGS	DWORD	[0x0, 0x1FF]	
• DECAYTIMESCALE	Flag bit		TRUE
• REFLECTIONSSCALE	Flag bit		TRUE
• REFLECTIONSDELAYSCALE	Flag bit		TRUE
• REVERBSCALE	Flag bit		TRUE
• REVERBDELAYSCALE	Flag bit		TRUE
• ECHOTIMESCALE	Flag bit		FALSE
• MODULATIONTIMESCALE	Flag bit		FALSE
• DECAYHFLIMIT	Flag bit		TRUE

EAXREVERB_ALLPARAMETERS

Parameter Name	EAXREVERB_ALLPARAMETERS
Parameter Type	EAXREVERBPROPERTIES structure

The EAXREVERBPROPERTIES structure is defined as: -

```
typedef struct _EAXREVERBPROPERTIES
{
  unsigned long ulEnvironment;
  float flEnvironmentSize;
  float flEnvironmentDiffusion;
  long lRoom;
  long lRoomHF;
  long lRoomLF;
  float flDecayTime;
  float flDecayHFRatio;
  float flDecayLFRatio;
  long lReflections;
  float flReflectionsDelay;
  EAXVECTOR vReflectionsPan;
  long lReverb;
  float flReverbDelay;
  EAXVECTOR vReverbPan;
  float flEchoTime;
  float flEchoDepth;
  float flModulationTime;
  float flModulationDepth;
  float flAirAbsorptionHF;
  float flHFReference;
  float flLFReference;
  float flRoomRolloffFactor;
  unsigned long ulFlags;
} EAXREVERBPROPERTIES, *LPEAXREVERBPROPERTIES;
```

EAXREVERB_ENVIRONMENT

Parameter Name	EAXREVERB_ENVIRONMENT
Parameter Type	ULONG
Default Value	EAX_ENVIRONMENT_GENERIC
Range	EAX_ENVIRONMENT_GENERIC to (EAX_ENVIRONMENT_COUNT – 1)
Units	Environment Enumeration value

The Environment property has been obsolete since EAX 3.0, and is supported for backwards code compatibility with EAX 2.0 applications. Using this property is not recommended when developing a new EAX 4.0 application. The EAXREVERB_ALLPARAMETERS property should be used instead for applying environment presets. The default value of the property is EAX_ENVIRONMENT_GENERIC, but its normal value is EAX_ENVIRONMENT_UNDEFINED.

Setting the Environment property to EAX_ENVIRONMENT_UNDEFINED has no effect on the other environment properties. Setting it to one of the legacy values enumerated in eax.h sets the values of all listener properties to reproduce the corresponding legacy EAX preset. If, subsequently, the value of one of the other listener properties is modified, the Environment property changes to EAX_ENVIRONMENT_UNDEFINED.

EAXREVERB_ENVIRONMENTSIZE

Parameter Name	EAXREVERB_ENVIRONMENTSIZE
Parameter Type	FLOAT
Default Value	7.5
Range	1.0 to 100.0
Units	Meters

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

Because Environment Size is a high-level property, when you change it, it can apply a relative adjustment to seven lower-level listener properties that determine the shape of the reverberation response: Reflections, Reflections Delay, Reverb, Reverb Delay, Decay Time, Echo Time, and Modulation Time.

The properties that will be affected are determined by the values of certain EAX Reverb flags described later on. There is a flag for each of the seven properties listed above, so the effect of Environment Size can be enabled / disabled for each of them. By default, only the first five flags are enabled (set to true), to provide a convincing simulation of change in the dimensions of a room (maintaining the reflective properties of its walls).

EAXREVERB_ENVIRONMENTDIFFUSION

Parameter Name	EAXREVERB_ENVIRONMENTDIFFUSION
Parameter Type	FLOAT
Default Value	1.0
Range	0.0 to 1.0
Units	A linear multiplier value

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

EAXREVERB_ROOM

Parameter Name	EAXREVERB_ROOM
Parameter Type	LONG
Default Value	-1000
Range	-10000 to 0
Units	mB (1/100th of a decibel)

The Room property is the master volume control for the reflected sound (both early reflections and reverberation) that EAX adds to all sound sources.

EAXREVERB_ROOMHF

Parameter Name	EAXREVERB_ROOMHF
Parameter Type	LONG
Default Value	-100
Range	-10000 to 0
Units	mB (1/100th of a decibel)

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

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

EAXREVERB_ROOMLF

Parameter Name	EAXREVERB_ROOMLF
Parameter Type	LONG
Default Value	0
Range	-10000 to 0
Units	mB (1/100th of a decibel)

The Room LF property further tweaks reflected sound by attenuating it at low frequencies. It controls a high-pass filter that applies globally to the reflected sound of all sound sources. The value of the Room LF property ranges from 0 dB (no filter) to -100 dB (virtually no reflected sound). It describes the relative attenuation obtained at the reference low frequency set by the property LF Reference.

Like the Room and Room HF properties, Room LF is a global parameter affecting all sound sources.

EAXREVERB_DECAYTIME

Parameter Name	EAXREVERB_DECAYTIME
Parameter Type	FLOAT
Default Value	1.49
Range	0.1 to 20.0
Units	Seconds

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

EAXREVERB_DECAYHFRATIO

Parameter Name	EAXREVERB_DECAYHFRATIO
Parameter Type	FLOAT
Default Value	0.83
Range	0.1 to 2.0
Units	A linear multiplier value

The Decay HF Ratio property sets the spectral quality of the reverberation decay. It is the ratio of high-frequency decay time—at the reference high frequency—relative to the time set by Decay Time. The Decay HF Ratio value 1.0 is neutral: the decay time is equal for high and mid frequencies.

As Decay HF Ratio increases above 1.0, the high frequency decay time increases so it is longer than the decay time at mid frequencies. You hear a more brilliant reverberation with a longer decay at high frequencies. As the Decay HF Ratio value decreases below 1.0, the high frequency decay time decreases so it is shorter than the decay time of the mid frequencies. You hear a more natural reverberation for a moderate setting, and a damper reverberation as you decrease the value further.

When you increase the Decay Time, the reverberation decay becomes proportionally longer at low and high frequencies. Since long decay times at high frequencies are unusual and may sound unnatural, EAX allows you to limit the high frequency decay time to a natural value by setting the flag Decay HF Limit, described below (this flag is set by default).

EAXREVERB_DECAYLFRATIO

Parameter Name	EAXREVERB_DECAYLFRATIO
Parameter Type	FLOAT
Default Value	1.0
Range	0.1 to 2.0
Units	A linear multiplier value

The Decay LF Ratio property sets the spectral quality of the reverberation decay. It is the ratio of low-frequency decay time—at the reference low frequency—relative to the time set by Decay Time. The Decay LF Ratio value 1.0 is neutral: the decay time is equal for low and mid frequencies.

As Decay LF Ratio increases above 1.0, the low frequency decay time increases so it is longer than the decay time at mid frequencies. As the Decay LF Ratio value decreases below 1.0, the low frequency decay time decreases so it is shorter than the decay time of the mid frequencies.

EAXREVERB_REFLECTIONS

Parameter Name	EAXREVERB_REFLECTIONS
Parameter Type	LONG
Default Value	-2602
Range	-10000 to 1000
Units	mB (1/100th of a decibel)

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

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

EAXREVERB_REFLECTIONSDELAY

Parameter Name	EAXREVERB_REFLECTIONSDELAY
Parameter Type	FLOAT
Default Value	0.007
Range	0.0 to 0.3
Units	Seconds

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

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

EAXREVERB_REFLECTIONSPAN

Parameter Name	EAXREVERB_REFLECTIONSPAN
Parameter Type	EAXVECTOR
Default Value	{0.0, 0.0, 0.0}
Range	Magnitude of vector from 0 to 1
Units	Dimensionless

The Reflections Pan property is a 3D vector that controls the spatial distribution of the cluster of early reflections. The direction of this vector controls the global direction of the reflections, while its magnitude controls how focused the reflections are towards this direction.

The direction of the vector is interpreted in the coordinate system of the user, without taking into account the orientation of the virtual listener. For instance, assuming a four-point loudspeaker playback system, setting Reflections Pan to (0., 0., 0.7) means that the reflections are panned to the front speaker pair, whereas a setting of (0., 0., −0.7) pans the reflections towards the rear speakers.

If the magnitude of Reflections Pan is zero (the default setting), the early reflections come evenly from all directions. As the magnitude increases, the reflections become more focused in the direction pointed to by the vector. A magnitude of 1.0 represents the extreme case where all reflections come from a single direction.

EAXREVERB_REVERB

Parameter Name	EAXREVERB_REVERB
Parameter Type	LONG
Default Value	200
Range	-10000 to 2000
Units	mB (1/100th of a decibel)

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

EAXREVERB_REVERBDELAY

Parameter Name	EAXREVERB_REVERBDELAY
Parameter Type	FLOAT
Default Value	0.011
Range	0.0 to 0.1
Units	Seconds

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

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

EAXREVERB_REVERBPAN

Parameter Name	EAXREVERB_REVERBPAN
Parameter Type	EAXVECTOR
Default Value	{0.0, 0.0, 0.0}
Range	Magnitude of vector from 0 to 1
Units	Dimensionless

The EAXREVERB_REVERBPAN property behaves in the same way as the *EAXREVERB_REFLECTIONSPAN* property, but applies to the late reverberation.

EAXREVERB_ECHOTIME

Parameter Name	EAXREVERB_ECHOTIME
Parameter Type	FLOAT
Default Value	0.25
Range	0.075 to 0.25
Units	Seconds

The property Echo Time determines the time lag of an echo in the reverberation decay. The strength of this effect is controlled by the settings of the Echo Depth and Environment Diffusion properties. For the maximum value of Diffusion, 1.0, the echo is restricted to the onset of the reverberation and perceptible mainly with percussive sounds. For lower values of Diffusion, the echo repeats cyclically along the reverberation decay, with a period equal to Echo Time. This creates an amplitude modulation in the decay, which washes off towards the end of the decay for intermediate settings of Diffusion. If Diffusion is minimal, the echo persists during the whole decay. The default value of Echo Time is 250 ms, which specifies a rate of 4 echoes per second.

EAXREVERB_ECHODEPTH

Parameter Name	EAXREVERB_ECHODEPTH
Parameter Type	FLOAT
Default Value	0.0
Range	0.0 to 1.0
Units	A linear multiplier value

Echo Depth is a factor controlling the strength of the echo effect.

EAXREVERB_MODULATIONTIME

Parameter Name	EAXREVERB_MODULATIONTIME
Parameter Type	FLOAT
Default Value	0.25
Range	0.04 to 4.0
Units	Seconds

The Modulation properties are similar to the Echo properties. However, instead of an echo (or amplitude modulation), they control a pitch modulation in the reverberation decay. The property Modulation Time controls the rate of the modulation. The default modulation time is 250 ms, which specifies a rate of 4 cycles per second. Unlike the Echo properties, the Modulation properties create a modulation that persists during the whole reverberation decay (irrespective of the setting of the Environment diffusion property).

EAXREVERB_MODULATIONDEPTH

Parameter Name	EAXREVERB_MODULATIONDEPTH
Parameter Type	FLOAT
Default Value	0.0
Range	0.0 to 1.0
Units	A linear multiplier value

The property Modulation Depth controls the pitch range of the modulation.

EAXREVERB_AIRABSORPTIONHF

Parameter Name	EAXREVERB_AIRABSORPTIONHF
Parameter Type	FLOAT
Default Value	-5.0
Range	-100.0 to 0.0
Units	mB / meter

The Air Absorption HF property controls the distance-dependent attenuation affecting the reverb at the reference high frequency caused by the propagation medium (while causing no attenuation at low frequencies). It also applies to the direct path if this EAX Reverb effect renders the listener environment (i.e. it is loaded in the Primary FX Slot and that slot is flagged as environmental). You can use Air Absorption HF to simulate sound transmission through foggy air, dry air, smoky atmosphere, and so on.

The Air Absorption HF property also determines the maximum value of the high-frequency decay time when the flag *EAXREVERBFLAGS_DECAYHFLIMIT* is set to TRUE.

The default value is -0.05 dB per meter, which roughly corresponds to typical conditions of atmospheric humidity, temperature, and so on. Lowering the value simulates a more absorbent medium (more humidity in the air, for example); raising the value simulates a less absorbent medium (dry desert air, for example).

EAXREVERB_HFREFERENCE

Parameter Name	EAXREVERB_HFREFERENCE
Parameter Type	FLOAT
Default Value	5000.0
Range	1000.0 to 20000.0
Units	Hertz

The HF Reference property determines the frequencies at which the high-frequency effects created by EAX properties are measured.

Note that, for EAX Reverb properties, it is necessary to maintain a factor of at least 10 between the low and high reference frequencies so that low frequency and high frequency properties can be accurately controlled and can produce independent effects. In other words, the LF Reference value should be less than 1/10 of the HF Reference value.

If the FX Slot hosting the EAX Reverb effect is flagged as environmental, the HF Reference value also applies to the effect send paths from all sources to this FX Slot. As a result, it is safe to adopt the same setting of HF Reference for all of the environment presets used in a given application. Otherwise, changing the Environment preset while the application is running will affect certain properties that should remain characteristics of the sources (such as their directivity at high frequencies, specified by the source property Outside Volume HF).

EAXREVERB_LFREFERENCE

Parameter Name	EAXREVERB_LFREFERENCE
Parameter Type	FLOAT
Default Value	250.0
Range	20.0 to 1000.0
Units	Hertz

The LF Reference property determines the frequencies at which the low-frequency effects created by EAX Reverb properties are measured.

Note that, it is necessary to maintain a factor of at least 10 between the low and high reference frequencies so that low frequency and high frequency properties can be accurately controlled and can produce independent effects. In other words, the LF Reference value should be less than 1/10 of the HF Reference value.

EAXREVERB_ROOMROLLOFFFACTOR

Parameter Name	EAXREVERB_ROOMROLLOFFFACTOR
Parameter Type	FLOAT
Default Value	0.0
Range	0.0 to 10.0
Units	A linear multiplier value

The Room Roll-Off Factor property is one of two methods available in EAX to attenuate the reflected sound (containing both reflections and reverberation) according to source-listener distance. It is defined the same way as OpenAL or DirectSound's Roll-Off Factor, but operates on reflected sound instead of direct-path sound. Setting the Room Roll-Off Factor value to 1.0 specifies that the reflected sound will decay by 6 dB every time the distance doubles. Any value other than 1.0 is equivalent to a scaling factor applied to the quantity specified by ((Source listener distance) - (Minimum Distance)). Minimum distance is a sound-source parameter that specifies the inner border for distance-related roll-off effects: if the source comes closer to the listener than the minimum distance, the sound is not increased as the source comes closer to the listener.

The default value of Room Roll-Off Factor is 0.0 because, by default, EAX naturally manages the reflected sound level automatically for each sound source to simulate the natural roll-off of reflected sound vs. distance in typical rooms. (Note that this is not the case if the source property flag Room Auto is set to false.) You can use Room Roll-Off Factor to exaggerate or replace the automatically controlled reverb roll-off.

EAXREVERB_FLAGS

Parameter Name	EAXREVERB_FLAGS
Parameter Type	ULONG
Default Value	EAXREVERBFLAGS_DECAYTIMESCALE | EAXREVERBFLAGS_REFLECTIONSSCALE | EAXREVERBFLAGS_REFLECTIONSDELAYSCALE | EAXREVERBFLAGS_REVERBSCALE | EAXREVERBFLAGS_REVERBDELAYSCALE | EAXREVERBFLAGS_DECAYHFLIMIT
Units	Each bit is a Boolean flag

The Flags property uses its eight low-order bits to set eight listener-property flags. Seven of these flags, the "scale flags," set the control that EAX Reverb property Environment Size has over seven lower-level listener properties. The eighth flag prevents excessive decay times at high frequencies, and is set to TRUE by default. The first five scale flags are also set to TRUE by default so Environment Size has control over the lower-level properties necessary to simulate the typical behavior expected for a change of room dimensions.

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

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

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

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

EAXREVERBFLAGS_REFLECTIONSSCALE
If both this flag and the Reflections Delay Scale flag are TRUE, an increase in Environment Size value causes an attenuation of the property Reflections. If one of the two flags is FALSE, a change in Environment Size has no effect on Reflections.

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

EAXREVERBFLAGS_ECHOTIMESCALE
If this flag is TRUE, a change in Environment Size value causes a proportional change of the property Echo Time. If it is FALSE (its default value), a change in Environment Size has no effect on Echo Time.

EAXREVERBFLAGS_MODULATIONTIMESCALE
If this flag is TRUE, a change in Environment Size value causes a proportional change of the property Modulation Time. If it is FALSE (its default value), a change in Environment Size has no effect on Modulation Time.

EAXREVERBFLAGS_DECAYHFLIMIT
If this flag is TRUE, the high-frequency decay time automatically stays below a limit value that is derived from the setting of the property Air Absorption HF. This maintains a natural sounding reverberation decay by allowing the application to increase the length of the Decay Time without the risk of obtaining an unnaturally long decay time at high frequencies. This limit applies regardless of the setting of the property Decay HF Ratio, and does not affect the value of Decay HF Ratio.

AGC Compressor Effect Properties

AGC Compressor property names are prefixed with EAXAGCCOMPRESSOR_

Properties	Type	Range	Default
ALLPARAMETERS	EAXAGCCOMPRESSORPROPERTIES		
ONOFF	ULONG	[0,1]	1

The Automatic Gain Control Compressor effect automatically evens out volume changes in an audio source. Quieter portions of the audio will be boosted while louder portions will stay the same or may even be reduced.

EAXAGCCOMPRESSOR_ALLPARAMETERS

Parameter Name	EAXAGCCOMPRESSOR_ALLPARAMETERS
Parameter Type	EAXAGCCOMPRESSORPROPERTIES structure

Allows setting all the AGC Compressor parameters with one call. Since this effect only has one property in EAX 4.0, that property can be set directly just as efficiently.

The EAXAGCCOMPRESSORPROPERTIES structure is defined as: -

```
typedef struct _EAXAGCCOMPRESSORPROPERTIES
{
  unsigned long ulOnOff;
} EAXAGCCOMPRESSORPROPERTIES, *LPEAXAGCCOMPRESSORPROPERTIES;
```

EAXAGCCOMPRESSOR_ONOFF

Parameter Name	EAXAGCCOMPRESSOR_ONOFF
Parameter Type	ULONG
Default Value	1
Range	0 (off) or 1 (on)

This property simply turns the compressor effect on or off.

Autowah Effect Properties

Autowah property names are prefixed with EAXAUTOWAH_

Properties	Type	Range	Default
ALLPARAMETERS	EAXAUTOWAHPROPERTIES		
ATTACKTIME	FLOAT	[0.0001,1.0]	0.06 secs
RELEASETIME	FLOAT	[0.0001,1.0]	0.06 secs
RESONANCE	LONG	[600,6000]	6000 mB
PEAKLEVEL	LONG	[-9000,9000]	2100 mB

The Autowah effect emulates the sound of a wah-wah pedal used with an electric guitar or a mute on a brass instrument. The Autowah creates the effect automatically according to the settings you choose. The effect is achieved by detecting the envelope of the input signal and applying a band-pass filter to the input.

EAXAUTOWAH_ALLPARAMETERS

Parameter Name	EAXAUTOWAH_ALLPARAMETERS
Parameter Type	EAXAUTOWAHPROPERTIES structure

Allows an application to set or get all the Autowah parameters with one call. The EAXAUTOWAHPROPERTIES structure is defined as: -

```
typedef struct _EAXAUTOWAHPROPERTIES
{
  float   flAttackTime;
  float   flReleaseTime;
  long    lResonance;
  long    lPeakLevel;
} EAXAUTOWAHPROPERTIES, *LPEAXAUTOWAHPROPERTIES;
```

EAXAUTOWAH_ATTACKTIME

Parameter Name	EAXAUTOWAH_ATTACKTIME
Parameter Type	FLOAT
Default Value	0.06
Range	0.0001 to 1.0
Units	Seconds

The rate at which the filter will sweep in response to a positive change in input signal volume.

EAXAUTOWAH_RELEASETIME

Parameter Name	EAXAUTOWAH_RELEASETIME
Parameter Type	FLOAT
Default Value	0.06
Range	0.0001 to 1.0
Units	Seconds

The time for the filter to return to its base centre frequency when the input signal stops.

EAXAUTOWAH_RESONANCE

Parameter Name	EAXAUTOWAH_RESONANCE
Parameter Type	LONG
Default Value	6000
Range	600 to 6000
Units	mB (1/100th of a decibel)

The height of the resonant peak of the band-pass filter. High values emphasize a narrow range of frequencies.

EAXAUTOWAH_PEAKLEVEL

Parameter Name	EAXAUTOWAH_PEAKLEVEL
Parameter Type	LONG
Default Value	2100
Range	-9000 to 9000
Units	mB (1/100th of a decibel)

The signal level at which the band-pass filter will attack to the maximum centre frequency. This adjusts the sensitivity of the envelope follower. In effect, high values lead to lower cut-off frequencies, and vice versa.

Chorus Effect Properties

Chorus property names are prefixed with EAXCHORUS_

Properties	Type	Range	Default
ALLPARAMETERS	EAXCHORUSPROPERTIES		
WAVEFORM	ULONG	[0,1]	1 (triangle)
PHASE	LONG	[-180,180]	90 degree
RATE	FLOAT	[0.0,10.0]	1.1 Hz
DEPTH	FLOAT	[0.0,1.0]	0.1
FEEDBACK	FLOAT	[-1.0,1.0]	0.25
DELAY	FLOAT	[0.0002,0.016]	0.016 secs

The Chorus effect can make a single instrument sound like several instruments. Chorus works by sampling a portion of the input signal, delaying it by a regularly varying time, then mixing it back with the source signal. The chorus effect is essentially the same affect as a flanger, though the flanger effect uses shorter delay times.

EAXCHORUS_ALLPARAMETERS

Parameter Name	EAXCHORUS_ALLPARAMETERS
Parameter Type	EAXCHORUSPROPERTIES structure

Allows setting all the Chorus parameters with one call. The EAXCHORUSPROPERTIES structure is defined as: -

```
typedef struct _EAXCHORUSPROPERTIES
{
  unsigned long   ulWaveform;
  long            lPhase;
  float           flRate;
  float           flDepth;
  float           flFeedback;
  float           flDelay;
} EAXCHORUSPROPERTIES, *LPEAXCHORUSPROPERTIES;
```

EAXCHORUS_WAVEFORM

Parameter Name	EAXCHORUS_WAVEFORM
Parameter Type	ULONG
Default Value	EAX_CHORUS_TRIANGLE
Range	EAX_CHORUS_TRIANGLE or EAX_CHORUS_SINUSOID
Units	Waveform Enumeration value

Selects the shape of the LFO waveform used to control the delay in the chorus.

EAXCHORUS_PHASE

Parameter Name	EAXCHORUS_PHASE
Parameter Type	LONG
Default Value	90
Range	-180 to 180
Units	Degrees

This changes the phase difference between the left and right LFO's. At zero degrees, the two LFOs are synchronized. Use this parameter to create the illusion of an expanded stereo field of the output signal.

EAXCHORUS_RATE

Parameter Name	EAXCHORUS_RATE
Parameter Type	FLOAT
Default Value	1.1
Range	0.0 to 10.0
Units	Hertz

Controls the rate of modulation.

EAXCHORUS_DEPTH

Parameter Name	EAXCHORUS_DEPTH
Parameter Type	FLOAT
Default Value	0.1
Range	0.0 to 1.0
Units	A linear multiplier value

This controls the amount by which the delay time is modulated by the LFO. Use this parameter to increase the pitch modulation. Large values will create a "warbling" effect.

EAXCHORUS_FEEDBACK

Parameter Name	EAXCHORUS_FEEDBACK
Parameter Type	FLOAT
Default Value	0.25
Range	-1.0 to 1.0

This controls the amount of the output signal fed back into the effect's input. A negative value will reverse the phase of the feedback signal. At full magnitude, the identical sample will repeat endlessly. At lower magnitudes, the sample will repeat and fade out over time. Use this parameter to create a "cascading" chorus effect.

EAXCHORUS_DELAY

Parameter Name	EAXCHORUS_DELAY
Parameter Type	FLOAT
Default Value	0.016
Range	0.0002 to 0.016
Units	Seconds

This is the average amount of time the sample is delayed before it is played back, and with feedback, the amount of time between iterations of the sample. Larger values lower the pitch. Smaller values make the chorus sound like a flanger, but with different frequency characteristics.

Distortion Effect Properties

Distortion property names are prefixed with EAXDISTORTION_

Properties	Type	Range	Default
ALLPARAMETERS	EAXDISTORSIONPROPERTIES		
EDGE	FLOAT	[0.0,1.0]	0.2
GAIN	LONG	[-6000,0]	-2600 mB
LOWPASSCUTOFF	FLOAT	[80.0,24000.0]	8000 Hz
EQCENTER	FLOAT	[80.0,24000.0]	3600 Hz
EQBANDWIDTH	FLOAT	[80.0,24000.0]	3600 Hz

The distortion effect simulates turning up (overdriving) the gain stage on a guitar amplifier or adding a distortion pedal to an instrument's output. It is achieved by clipping the signal (adding more square wave-like components) and adding rich harmonics.

EAXDISTORTION_ALLPARAMETERS

Parameter Name	EAXDISTORTION_ALLPARAMETERS
Parameter Type	EAXDISTORTIONPROPERTIES structure

Allows setting all the Distortion parameters with one call. The EAXDISTORTIONPROPERTIES structure is defined as: -

```
typedef struct _EAXDISTORTIONPROPERTIES
{
  float   flEdge;
  long    lGain;
  float   flLowPassCutOff;
  float   flEQCenter;
  float   flEQBandwidth;
} EAXDISTORTIONPROPERTIES, *LPEAXDISTORTIONPROPERTIES;
```

EAXDISTORTION_EDGE

Parameter Name	EAXDISTORTION_EDGE
Parameter Type	FLOAT
Default Value	0.2
Range	0.0 to 1.0

This sets the distortion's intensity. High values increase the amount of "fuzz" in the signal.

EAXDISTORTION_GAIN

Parameter Name	EAXDISTORTION_GAIN
Parameter Type	LONG
Default Value	-2600
Range	-6000 to 0
Units	mB (1/100th of a decibel)

This decreases the signal level after the signal has been distorted.

EAXDISTORTION_LOWPASSCUTOFF

Parameter Name	EAXDISTORTION_LOWPASSCUTOFF
Parameter Type	FLOAT
Default Value	8000
Range	80.0 to 24000.0
Units	Hertz

This reduces the number of higher harmonics introduced by the distortion.

EAXDISTORTION_EQCENTER

Parameter Name	EAXDISTORTION_EQCENTER
Parameter Type	FLOAT
Default Value	3600
Range	80.0 to 24000.0
Units	Hertz

Sets the centre frequency of the range of harmonic content added to the signal.

EAXDISTORTION_EQBANDWIDTH

Parameter Name	EAXDISTORTION_EQBANDWIDTH
Parameter Type	FLOAT
Default Value	3600
Range	80.0 to 24000.0
Units	Hertz

This is the width of the frequency band that determines the range of the harmonic content added to the signal.

Echo Effect Properties

Echo property names are prefixed with EAXECHO_

Properties	Type	Range	Default
ALLPARAMETERS	EAXECHOPROPERTIES		
DELAY	FLOAT	[0.002,0.207]	0.1 secs
LRDELAY	FLOAT	[0,0.404]	0.1 secs
DAMPING	FLOAT	[0.0,0.99]	0.5
FEEDBACK	FLOAT	[0.0,1.0]	0.5
SPREAD	FLOAT	[-1.0,1.0]	-1.0

The echo effect is used to bring movement and spatial expansion to a source. It is achieved by sampling the input signal and then replaying the sample after a programmable delay. As opposed to reverberation, this effect creates one or more discrete echoes.

EAXECHO_ALLPARAMETERS

Parameter Name	EAXECHO_ALLPARAMETERS
Parameter Type	EAXECHOPROPERTIES structure

Allows setting all the Echo parameters with one call. The EAXECHOPROPERTIES structure is defined as: -

```
typedef struct _EAXECHOPROPERTIES
{
  float   flDelay;
  float   flLRDelay;
  float   flDamping;
  float   flFeedback;
  float   flSpread;
} EAXECHOPROPERTIES, *LPEAXECHOPROPERTIES;
```

EAXECHO_DELAY

Parameter Name	EAXECHO_DELAY
Parameter Type	FLOAT
Default Value	0.1
Range	0.002 to 0.207
Units	Seconds

The amount of time between when the input is sampled and replayed.

EAXECHO_LRDELAY

Parameter Name	EAXECHO_LRDELAY
Parameter Type	FLOAT
Default Value	0.1
Range	0.0 to 0.404
Units	Seconds

Controls the delay time between the first and second echo taps.

EAXECHO_DAMPING

Parameter Name	EAXECHO_DAMPING
Parameter Type	FLOAT
Default Value	0.5
Range	0.0 to 0.99

Controls the amount by which each echo is damped.

EAXECHO_FEEDBACK

Parameter Name	EAXECHO_FEEDBACK
Parameter Type	FLOAT
Default Value	0.5
Range	0.0 to 1.0

The amount of the output signal fed back into the input. Use this parameter to create “cascading” echoes. At full magnitude, the identical sample will repeat endlessly. Below full magnitude, the sample will repeat and fade.

EAXECHO_SPREAD

Parameter Name	EAXECHO_SPREAD
Parameter Type	FLOAT
Default Value	-1.0
Range	-1.0 to 1.0

Controls the left right spread of the echoes.

Equalizer Effect Properties

Equalizer property names are prefixed with EAXEQUALIZER_

Properties	Type	Range	Default
ALLPARAMETERS	EAXEQUALIZERPROPERTIES		
LOWGAIN	LONG	[-1800,1800]	0 mB
LOWCUTOFF	FLOAT	[50.0,800.0]	200 Hz
MID1GAIN	LONG	[-1800,1800]	0 mB
MID1CENTER	FLOAT	[200.0,3000.0]	500 Hz
MID1WIDTH	FLOAT	[0.01,1.0]	1.0 octave
MID2GAIN	LONG	[-1800,1800]	0 mB
MID2CENTER	FLOAT	[1000.0,8000.0]	3000 Hz
MID2WIDTH	FLOAT	[0.01,1.0]	1.0 octave
HIGHGAIN	LONG	[-1800,1800]	0 mB
HIGHCUTOFF	FLOAT	[4000.0,16000.0]	6000 Hz

The equalizer effect can boost or cut the input signal in four programmable frequency ranges. The lowest frequency range is called “low.” The middle ranges are called “mid1” and “mid2.” The high range is called “high.”

EAXEQUALIZER_ALLPARAMETERS

Parameter Name	EAXEQUALIZER_ALLPARAMETERS
Parameter Type	EAXEQUALIZERPROPERTIES structure

Allows setting all the Equalizer parameters with one call. The EAXEQUALIZERPROPERTIES structure is defined as: -

```
typedef struct _EAXEQUALIZERPROPERTIES
{
  long    lLowGain;
  float   flLowCutOff;
  long    lMid1Gain;
  float   flMid1Center;
  float   flMid1Width;
  long    lMid2Gain;
  float   flMid2Center;
  float   flMid2Width;
  long    lHighGain;
  float   flHighCutOff;
} EAXEQUALIZERPROPERTIES, *LPEAXEQUALIZERPROPERTIES;
```

EAXEQUALIZER_LOWGAIN

Parameter Name	EAXEQUALIZER_LOWGAIN
Parameter Type	LONG
Default Value	0
Range	-1800 to 1800
Units	mB (1/100th of a decibel)

The amount that the low frequencies should be cut or boosted.

EAXEQUALIZER_LOWCUTOFF

Parameter Name	EAXEQUALIZER_LOWCUTOFF
Parameter Type	FLOAT
Default Value	200
Range	50.0 to 800.0
Units	Hertz

The low frequency cut-off value for the low pass filter component of the equalizer.

EAXEQUALIZER_MID1GAIN

Parameter Name	EAXEQUALIZER_MID1GAIN
Parameter Type	LONG
Default Value	0
Range	-1800 to 1800
Units	mB (1/100th of a decibel)

The amount that the first (of two) middle frequencies should be cut or boosted.

EAXEQUALIZER_MID1CENTER

Parameter Name	EAXEQUALIZER_MID1CENTER
Parameter Type	FLOAT
Default Value	500
Range	200 to 3000
Units	Hertz

The centre frequency for the “mid1” filter component of the equalizer.

EAXEQUALIZER_ MID1WIDTH

Parameter Name	EAXEQUALIZER_MID1WIDTH
Parameter Type	FLOAT
Default Value	1.0
Range	0.01 to 1.0
Units	Octaves

The width of the “mid1” filter component of the equalizer.

EAXEQUALIZER_MID2GAIN

Parameter Name	EAXEQUALIZER_MID2GAIN
Parameter Type	LONG
Default Value	0
Range	-1800 to 1800
Units	mB (1/100th of a decibel)

The amount that the second (of two) middle frequencies should be cut or boosted.

EAXEQUALIZER_MID2CENTER

Parameter Name	EAXEQUALIZER_MID2CENTER
Parameter Type	FLOAT
Default Value	3000
Range	1000 to 8000
Units	Hertz

The centre frequency for the “mid2” filter component of the equalizer.

EAXEQUALIZER_MID2WIDTH

Parameter Name	EAXEQUALIZER_MID2WIDTH
Parameter Type	FLOAT
Default Value	1.0
Range	0.01 to 1.0
Units	Octaves

The width of the “mid2” component of the equalizer.

EAXEQUALIZER_HIGHGAIN

Parameter Name	EAXEQUALIZER_HIGHGAIN
Parameter Type	LONG
Default Value	0
Range	-1800 to 1800
Units	mB (1/100th of a decibel)

The amount that the high frequencies should be cut or boosted.

EAXEQUALIZER_HIGHCUTOFF

Parameter Name	EAXEQUALIZER_HIGHCUTOFF
Parameter Type	FLOAT
Default Value	6000
Range	4000 to 16000
Units	Hertz

The cut-off frequency for the high pass filter component of the equalizer.

Flanger Effect Properties

Flanger property names are prefixed with EAXFLANGER_

Properties	Type	Range	Default
ALLPARAMETERS	EAXFLANGERPROPERTIES		
WAVEFORM	ULONG	[0,1]	1 (triangle)
PHASE	LONG	[-180,180]	0 degree
RATE	FLOAT	[0.0,10.0]	0.27 Hz
DEPTH	FLOAT	[0.0,1.0]	1.0
FEEDBACK	FLOAT	[-1.0,1.0]	-0.5
DELAY	FLOAT	[0.0002,0.004]	0.002 secs

The flanger effect creates a “tunnelling” or “whooshing” sound (like a jet flying overhead). It works by sampling a portion of the input signal, delaying it by a period modulated between 0 and 4ms by a low-frequency oscillator, and then mixing it with the source signal.

EAXFLANGER_ALLPARAMETERS

Parameter Name	EAXFLANGER_ALLPARAMETERS
Parameter Type	EAXFLANGERPROPERTIES structure

Allows setting all the Flanger parameters with one call. The EAXFLANGERPROPERTIES structure is defined as: -

```
typedef struct _EAXFLANGERPROPERTIES
{
  unsigned long   ulWaveform;
  long            lPhase;
  float           flRate;
  float           flDepth;
  float           flFeedback;
  float           flDelay;
} EAXFLANGERPROPERTIES, *LPEAXFLANGERPROPERTIES;
```

EAXFLANGER_WAVEFORM

Parameter Name	EAXFLANGER_WAVEFORM
Parameter Type	ULONG
Default Value	EAX_FLANGER_TRIANGLE
Range	EAX_FLANGER_TRIANGLE or EAX_FLANGER_SINUSOID
Units	Waveform Enumeration value

Selects the shape of the LFO waveform that represents the amount of the delay of the sampled signal. Zero is a sinusoid and one is a triangle.

EAXFLANGER_PHASE

Parameter Name	EAXFLANGER_PHASE
Parameter Type	LONG
Default Value	0
Range	-180 to 180
Units	Degrees

This changes the phase difference between the left and right LFO's. At zero degrees, the two LFOs are synchronized.

EAXFLANGER_RATE

Parameter Name	EAXFLANGER_RATE
Parameter Type	FLOAT
Default Value	0.27
Range	0.0 to 10.0
Units	Hertz

The number of times per second the LFO controlling the amount of delay repeats. Higher values increase the pitch modulation.

EAXFLANGER_DEPTH

Parameter Name	EAXFLANGER_DEPTH
Parameter Type	FLOAT
Default Value	1.0
Range	0.0 to 1.0

The ratio by which the delay time is modulated by the LFO. Use this parameter to increase the pitch modulation.

EAXFLANGER_FEEDBACK

Parameter Name	EAXFLANGER_FEEDBACK
Parameter Type	FLOAT
Default Value	-0.5
Range	-1.0 to 1.0

This is the amount of the output signal level fed back into the effect's input. A negative value will reverse the phase of the feedback signal. Use this parameter to create an "intense metallic" effect.

At full magnitude, the identical sample will repeat endlessly. At less than full magnitude, the sample will repeat and fade out over time.

EAXFLANGER_DELAY

Parameter Name	EAXFLANGER_DELAY
Parameter Type	FLOAT
Default Value	0.002
Range	0.0002 to 0.004
Units	Seconds

The average amount of time the sample is delayed before it is played back; with feedback, the amount of time between iterations of the sample.

Frequency Shifter Properties

Frequency Shifter property names are prefixed with EAXFREQUENCYSHIFTER_

Properties	Type	Range	Default
ALLPARAMETERS	EAXFREQUENCYSHIFTERPROPERTIES		
FREQUENCY	FLOAT	[0.0,24000.0]	0 Hz
LEFTDIRECTION	ULONG	[0,2]	0 (down)
RIGHTDIRECTION	ULONG	[0,2]	0 (down)

This effect shifts the frequency of the input signal in real time. Unlike the pitch shifter, harmonic relationships are not maintained.

EAXFREQUENCYSHIFTER_ALLPARAMETERS

Parameter Name	EAXFREQUENCYSHIFTER_ALLPARAMETERS
Parameter Type	EAXFREQUENCYSHIFTERPROPERTIES structure

Allows setting all the Frequency Shifter parameters with one call. The EAXFREQUENCYSHIFTERPROPERTIES structure is defined as: -

```
typedef struct _EAXFREQUENCYSHIFTERPROPERTIES
{
  float           flFrequency;
  unsigned long   ulLeftDirection;
  unsigned long   ulRightDirection;
} EAXFREQUENCYSHIFTERPROPERTIES,
*LPEAXFREQUENCYSHIFTERPROPERTIES;
```

EAXFREQUENCYSHIFTER_FREQUENCY

Parameter Name	EAXFREQUENCYSHIFTER_FREQUENCY
Parameter Type	FLOAT
Default Value	0
Range	0.0 to 24000.0
Units	Hertz

This is the carrier frequency. For carrier frequencies below the audible range, the single-sideband modulator may produce phaser effects, spatial effects or a slight pitch-shift. As the carrier frequency increases, the timbre of the sound is affected; a piano or guitar note becomes like a bell's chime, and a human voice sounds extraterrestrial.

EAXFREQUENCYSHIFTER_LEFTDIRECTION

Parameter Name	EAXFREQUENCYSHIFTER_LEFTDIRECTION
Parameter Type	ULONG
Default Value	EAX_FREQUENCYSHIFTER_DOWN
Range	EAX_FREQUENCYSHIFTER_DOWN or EAX_FREQUENCYSHIFTER_UP or EAX_FREQUENCYSHIFTER_OFF
Units	Direction enumeration value

These select which internal signals are added together to produce the output. Different combinations of values will produce slightly different tonal and spatial effects.

EAXFREQUENCYSHIFTER_RIGHTDIRECTION

Parameter Name	EAXFREQUENCYSHIFTER_RIGHTDIRECTION
Parameter Type	ULONG
Default Value	EAX_FREQUENCYSHIFTER_DOWN
Range	EAX_FREQUENCYSHIFTER_DOWN or EAX_FREQUENCYSHIFTER_UP or EAX_FREQUENCYSHIFTER_OFF
Units	Direction enumeration value

These select which internal signals are added together to produce the output. Different combinations of values will produce slightly different tonal and spatial effects.

Vocal Morpher Properties

Vocal Morpher property names are prefixed with EAXVOCALMORPHER_

Properties	Type	Range	Default
ALLPARAMETERS	EAXVOCALMORPHERPROPERTIES		
PHONEMEA	ULONG	[0,29]	0 (A)
PHONEMEACOARSETUNING	LONG	[-24,24]	0 semitone
PHONEMEB	ULONG	[0,29]	10 (ER)
PHONEMEBCOARSETUNING	LONG	[-24,24]	0 semitone
WAVEFORM	ULONG	[0,2]	0 (sinusoid)
RATE	FLOAT	[0.0,10.0]	1.41 Hz

The Vocal Morpher consists of a pair of 4-band formant filters, used to impose vocal tract effects upon the input signal. If the input signal is a broadband sound such as pink noise or a car engine, the Vocal Morpher can provide a wide variety of filtering effects. A low-frequency oscillator can be used to morph the filtering effect between two different phonemes. The Vocal Morpher is not necessarily intended for use on voice signals; it is primarily intended for pitched noise effects, vocal-like wind effects, etc.

EAXVOCALMORPHER_ALLPARAMETERS

Parameter Name	EAXVOCALMORPHER_ALLPARAMETERS
Parameter Type	EAXVOCALMORPHERPROPERTIES structure

Allows setting all the Vocal Morpher parameters with one call. The EAXVOCALMORPHERPROPERTIES structure is defined as: -

```
typedef struct _EAXVOCALMORPHERPROPERTIES
{
  unsigned long   ulPhonemeA;
  long            lPhonemeACoarseTuning;
  unsigned long   ulPhonemeB;
  long            lPhonemeBCoarseTuning;
  unsigned long   ulWaveform;
  float           flRate;
} EAXVOCALMORPHERPROPERTIES, *LPEAXVOCALMORPHERPROPERTIES;
```

EAXVOCALMORPHER_PHONEMEA

Parameter Name	EAXVOCALMORPHER_PHONEMEA
Parameter Type	ULONG
Default Value	A
Range	One of these phonemes: - A, E, I, O, U, AA, AE, AH, AO, EH, ER, IH, IY, UH, UW, B, D, F,G, J, K, L, M, N, P, R, S, T, V, Z
Units	Enumeration value

If Phoneme A and Phoneme B are set to the same value, that determines the filtering effect that will be heard. If these two parameters are set to different phonemes, the filtering effect will morph between the two settings at a rate specified by *EAXVOCALMORPHER_RATE*.

EAXVOCALMORPHER_PHONEMEACOARSETUNING

Parameter Name	EAXVOCALMORPHER_PHONEMEACOARSETUNING
Parameter Type	LONG
Default Value	0
Range	-24 to 24
Units	Semi-tones

These are used to adjust the pitch of the phoneme filters in 1 semitone increments.

EAXVOCALMORPHER_PHONEMEB

Parameter Name	EAXVOCALMORPHER_PHONEMEB
Parameter Type	ULONG
Default Value	ER
Range	One of these phonemes: - A, E, I, O, U, AA, AE, AH, AO, EH, ER, IH, IY, UH, UW, B, D, F,G, J, K, L, M, N, P, R, S, T, V, Z
Units	Enumeration value

If Phoneme A and Phoneme B are set to the same value, that determines the filtering effect that will be heard. If these two parameters are set to different phonemes, the filtering effect will morph between the two settings at a rate specified by *EAXVOCALMORPHER_RATE*.

EAXVOCALMORPHER_PHONEMEBCOARSETUNING

Parameter Name	EAXVOCALMORPHER_PHONEMEBCOARSETUNING
Parameter Type	LONG
Default Value	0
Range	-24 to 24
Units	Semi-tones

These are used to adjust the pitch of the phoneme filters in 1 semi-tone increments.

EAXVOCALMORPHER_WAVEFORM

Parameter Name	EAXVOCALMORPHER_WAVEFORM
Parameter Type	ULONG
Default Value	EAX_VOCALMORPHER_SINUSOID
Range	EAX_VOCALMORPHER_SINUSOID or EAX_VOCALMORPHER_TRIANGLE or EAX_VOCALMORPHER_SAWTOOTH
Units	Waveform enumeration

This controls the shape of the low-frequency oscillator used to morph between the two phoneme filters.

By selecting a saw-tooth wave and a slow *EAXVOCALMORPHER_RATE*, one can create a filtering effect that slowly increases or decreases in pitch (depending on which of the two phoneme filters A or B is perceived as being higher-pitched).

EAXVOCALMORPHER_RATE

Parameter Name	EAXVOCALMORPHER_RATE
Parameter Type	FLOAT
Default Value	1.41
Range	0 to 10
Units	Hertz

This controls the frequency of the low-frequency oscillator used to morph between the two phoneme filters.

Pitch Shifter Properties

Pitch Shifter property names are prefixed with EAXPITCHSHIFTER_

Properties	Type	Range	Default
ALLPARAMETERS	EAXPITCHSHIFTERPROPERTIES		
COARSETUNE	LONG	[-12,12]	12 semitone
FINETUNE	LONG	[-50,50]	0 cent

This effect shifts the frequency of the input signal in real time, while preserving harmonic relationships. The effect is achieved by doing sample-rate conversion on the fly.

EAXPITCHSHIFTER_ALLPARAMETERS

Parameter Name	EAXPITCHSHIFTER_ALLPARAMETERS
Parameter Type	EAXPITCHSHIFTERPROPERTIES structure

Allows setting all the Pitch Shifter parameters with one call. The EAXPITCHSHIFTERPROPERTIES structure is defined as: -

```
typedef struct _EAXPITCHSHIFTERPROPERTIES
{
  long    lCoarseTune;
  long    lFineTune;
} EAXPITCHSHIFTERPROPERTIES, *LPEAXPITCHSHIFTERPROPERTIES;
```

EAXPITCHSHIFTER_COARSETUNE

Parameter Name	EAXPITCHSHIFTER_COARSETUNE
Parameter Type	LONG
Default Value	12
Range	-12 to 12
Units	Semitones

This sets the number of semi-tones by which the pitch is shifted. There are 12 semitones per octave. Negative values decrease the amount of the shift, and positive values increase it.

EAXPITCHSHIFTER_FINETUNE

Parameter Name	EAXPITCHSHIFTER_FINETUNE
Parameter Type	LONG
Default Value	0
Range	-50 to 50
Units	Cents

This sets the number of cents between semi-tones that a pitch is shifted. A cent is 1/100th of a semitone. Negative values decrease the amount of the shift and positive values increase it.

Ring Modulator Properties

Ring Modulator property names are prefixed with EAXRINGMODULATOR_

Properties	Type	Range	Default
ALLPARAMETERS	EAXRINGMODULATORPROPERTIES		
FREQUENCY	FLOAT	[0.0,8000.0]	440 Hz
HIGHPASSCUTOFF	FLOAT	[0.0,24000.0]	800 Hz
WAVEFORM	ULONG	[0,2]	0 (sinusoid)

The ring modulator multiplies an input signal by a carrier signal in the time domain, resulting in tremolo or in-harmonic effects.

EAXRINGMODULATOR_ALLPARAMETERS

Parameter Name	EAXRINGMODULATOR_ALLPARAMETERS
Parameter Type	EAXRINGMODULATORPROPERTIES structure

Allows setting all the Ring Modulator parameters with one call. The EAXRINGMODULATORPROPERTIES structure is defined as: -

```
typedef struct _EAXRINGMODULATORPROPERTIES
{
  float           flFrequency;
  float           flHighPassCutOff;
  unsigned long   ulWaveform;
} EAXRINGMODULATORPROPERTIES, *LPEAXRINGMODULATORPROPERTIES;
```

EAXRINGMODULATOR_FREQUENCY

Parameter Name	EAXRINGMODULATOR_FREQUENCY
Parameter Type	FLOAT
Default Value	440
Range	0 to 8000
Units	Hertz

This is the frequency of the carrier signal. If the carrier signal is slowly varying (less than 20 Hz), the result is a tremolo (slow amplitude variation) effect. If the carrier signal is in the audio range, audible upper and lower sidebands begin to appear, causing an in-harmonic effect. The carrier signal itself is not heard in the output.

EAXRINGMODULATOR_HIGHPASSCUTOFF

Parameter Name	EAXRINGMODULATOR_HIGHPASSCUTOFF
Parameter Type	FLOAT
Default Value	800
Range	0 to 24000
Units	Hertz

This controls the cut-off frequency at which the input signal is high-pass filtered before being ring modulated. If the cut-off frequency is 0, the entire signal will be ring modulated. If the cut-off frequency is high, very little of the signal (only those parts above the cut-off) will be ring modulated.

EAXRINGMODULATOR_WAVEFORM

Parameter Name	EAXRINGMODULATOR_WAVEFORM
Parameter Type	ULONG
Default Value	EAX_RINGMODULATOR_SINUSOID
Range	EAX_RINGMODULATOR_SINUSOID or EAX_RINGMODULATOR_SAWTOOTH or EAX_RINGMODULATOR_SQUARE
Units	Waveform enumeration value

This controls which waveform is used as the carrier signal.

Traditional ring modulator and tremolo effects generally use a sinusoidal carrier. Saw-tooth and square waveforms are likely to cause unpleasant aliasing.