

3DxpTM

DirectSound 3.0 Extension API

**Prepared by the
Interactive Audio Special Interest Group
3D Working Group**

June 1, 1997

Version 1.0

Recommended by:

Aural Semiconductor, Creative Labs, Diamond Multimedia, DiamondWare, Gulbransen,
Spatializer Audio Labs, Texas Instruments, VLSI Technology, S3, Q-Sound Labs,
Rockwell Semiconductor

The MMA, IASIG, and all members shall not be held liable to any person or entity for any reason related to the adoption, implementation or other use of this document or the specification herein.

This document was prepared by a group of 3D hardware and software vendors meeting as the 3D Audio Working Group (3DWG) of the Interactive Audio Special Interest Group (IASIG) of the MIDI Manufacturers Association.

It is intended as an alternative for developers who wish to extend or accelerate the functionality of Microsoft's DirectSound 3.0 API (for Windows applications) in a standard fashion prior to the release of a Microsoft solution, a desire which is supported by members of the 3DWG. This recommendation is not endorsed by Microsoft.

The 3DWG has also prepared guidelines – specifically a “Level 1” and a “Level 2” document - which define more fully the functions and parameters which the WG believe are most appropriate for accurate 3D object and environment modeling, and are not OS or API specific. These guidelines should help move the entire PC industry in a common direction, towards greater quality and superior performance, and more realistic multimedia applications.

3Dxp is a trademark of MIDI Manufacturers Association Incorporated. All other trademarks are property of their holders.

Copyright © 1997 MIDI Manufacturers Association Incorporated

ALL RIGHTS RESERVED. NO PART OF THIS DOCUMENT MAY BE REPRODUCED IN ANY FORM OR BY ANY MEANS, ELECTRONIC OR MECHANICAL, INCLUDING INFORMATION STORAGE AND RETRIEVAL SYSTEMS, WITHOUT PERMISSION IN WRITING FROM THE MIDI MANUFACTURERS ASSOCIATION.

MMA
POB 3173
La Habra CA 90632-3173

Table of Contents

| | |
|------------------------------------|----|
| Introduction | 1 |
| Abstract:..... | 1 |
| The Problem: | 1 |
| The Solution: | 1 |
| Objectives:..... | 1 |
| Overview | 2 |
| The Parameters to be passed: | 2 |
| Listener and Source..... | 2 |
| Source only | 2 |
| 3Dxp Operation Description:..... | 2 |
| The Game/Application layer:..... | 3 |
| The 3Dxp extension layer:..... | 4 |
| 3Dxp Reference Code Path:..... | 5 |
| Game/Application Guidelines..... | 6 |
| The Theory: | 6 |
| The Details: | 6 |
| Declarations for 3Dxp:..... | 6 |
| Querying a 3Dxp Driver:..... | 6 |
| Localizing a Sound:..... | 7 |
| Reference Code:..... | 8 |
| At a Minimum: | 8 |
| How It Works (briefly):..... | 8 |
| Errata/App Notes: | 8 |
| Bugs: | 8 |
| C vs. C++ | 8 |
| Potential Problems: | 9 |
| Driver Guidelines..... | 10 |
| 3Dxp Driver Sample Code: | 10 |
| 3Dxp Driver Lock Sequence:..... | 12 |
| 3Dxp Driver Unlock Sequence: | 13 |

Introduction

Abstract:

The new generation of audio hardware supporting 3D localization can provide startling positional accuracy. It is now possible for listeners to perceive sounds emanating from above, below and even behind them, regardless of source (speaker) location.

This document describes a recommended “universal” approach for software (Windows/Intel) to support multi-vendor solutions and Microsoft DirectSound3D with a single API.

Microsoft has already announced revisions to DirectSound (“DirectX5”) which will also provide API extensions for hardware acceleration, but for those software developers who desire an alternative solution, 3Dxp has been created and made available by the IASIG 3DWG.

The Problem:

As an API, Microsoft’s DirectSound3D goes a long way towards addressing the issues of 3D audio. However, DirectSound3D (version 3.0) does not pass 3D parameters to external hardware or software, and implements only one version of HRTF processing. This method of coding locks out third party solutions – many of which could decrease the load on the host CPU and offer far superior positioning quality. Though Microsoft is aware of this and has promised a solution in a later release of DirectSound, many IHVs and ISVs may prefer an immediate solution. However, at the same time, game developers can not justify support for multiple methods of 3D audio.

The Solution:

3Dxp was created as a single API that all 3D hardware/software vendors could use for extending DirectSound and significantly improving the 3D experience of PC games and multimedia applications under Windows. 3Dxp is a simple layer added to DirectSound3D and to the Game/Application which enables parameter passing to external hardware or software. 3Dxp does not attempt to reinvent the terminology, paradigm, 3D audio model, or data structures of DirectSound3D. Rather, 3Dxp’s structures and model are 100% backward-compatible with DirectSound3D.

Objectives:

- Enable DirectSound3D accelerators
- Provide full DirectSound3D functionality
- Keep DirectSound3D parameters and data structures
- Keep it simple
- Allow for future extensibility
- Offer without license or restriction

Overview

The Parameters to be passed:

DirectSound3D (version 3.0) unmodified parameters:

Listener and Source

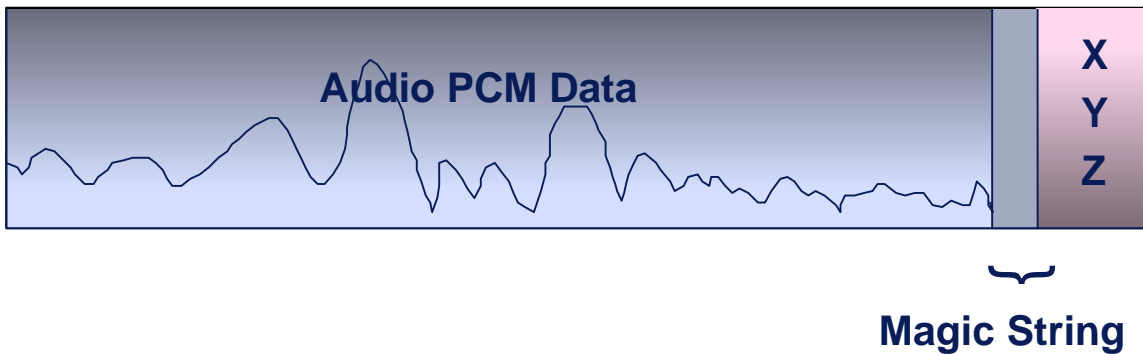
- vPosition (x, y, z)
- vVelocity
- Listener only
- vOrientFront, vOrientTop
- flDistanceFactor
- flRolloffFactor
- flDopplerFactor

Source only

- dwInsideConeAngle
- dwOutsideConeAngle
- vConeOrientation
- lConeOutsideVolume
- flMinDistance
- flMaxDistance

3Dxp Operation Description:

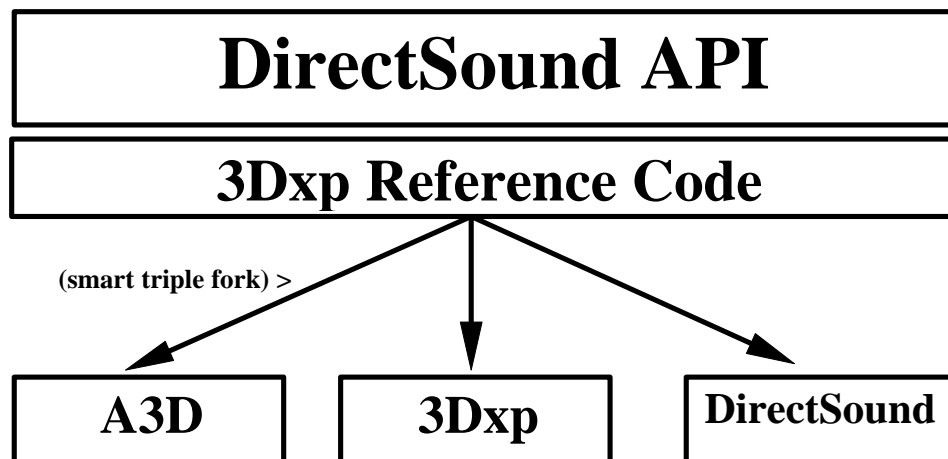
- Append 3D info to DirectSound buffer
- Sixteen byte ID plus 132 bytes of info (magic string + xyz)
- App determines if driver is compliant
- Create special secondary buffer
- Write magic query String
- Read back confirmation of compliance



The Game/Application layer:

3Dxp works by adding simple layer to DirectSound3D and to the Game/Application which enables parameter passing to external hardware or software. 3Dxp is sufficiently similar to DirectSound3D that it's easily possible to make a wrapper which goes through 3Dxp or A3D (see note below) if installed, or otherwise uses DirectSound3D. Such a wrapper is provided as the "3Dxp Reference Code", included in the 3Dxp SDK.

- Reference Code Auto Detects device
- 3Dxp, A3D or Standard DirectSound3D is used



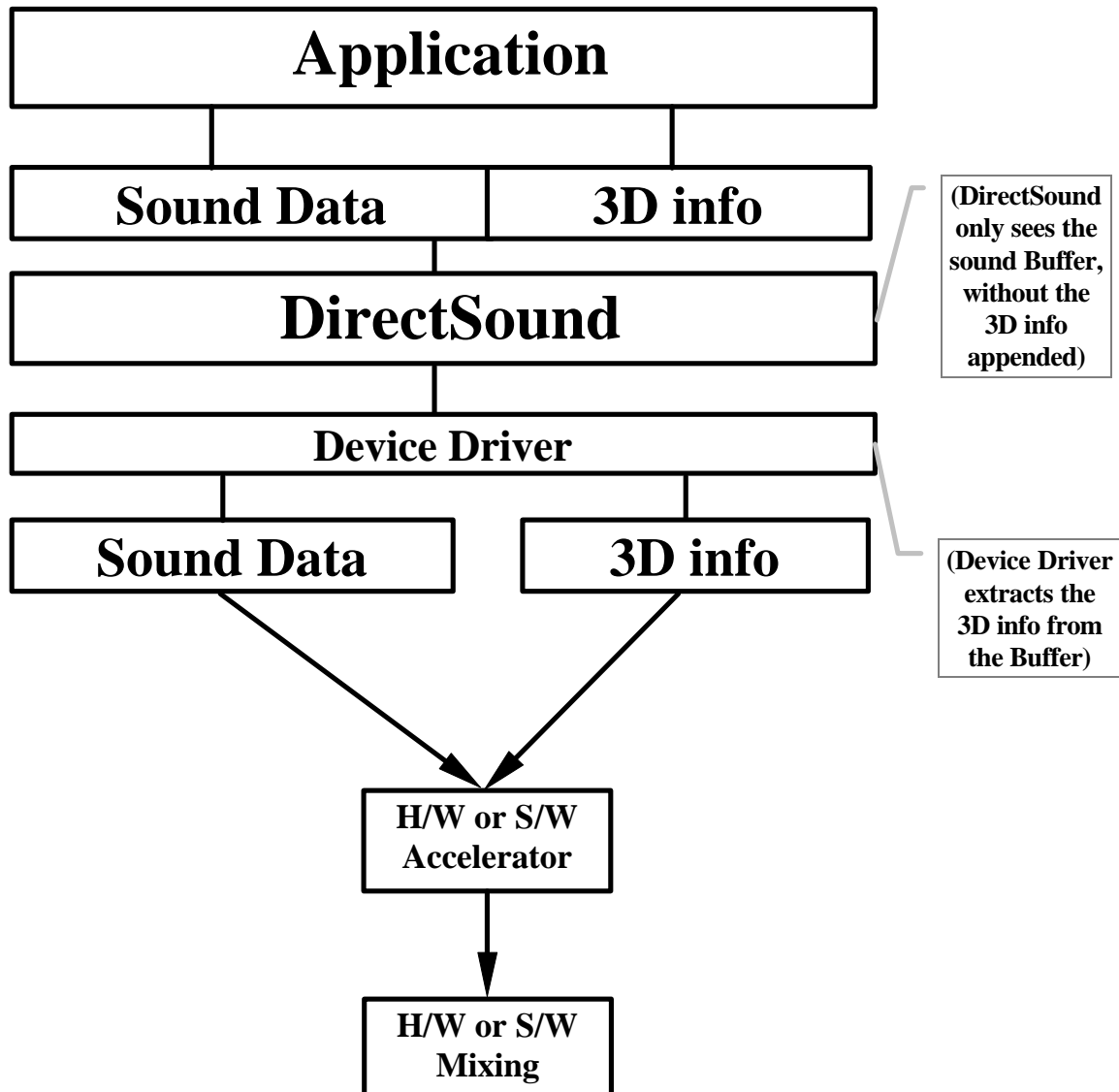
Note: The above represents a common 3Dxp Reference Code which can utilize 3Dxp applications, A3D applications or fall-back to DirectSound3D on the host.

Note on A3D: A3D is a 3D audio renderer from Aureal Semiconductor that has been licensed to several multimedia vendors. Please check www.a3d.com for more information on the A3D features. Please note that A3D is only supported in the 3Dxp guideline when using the reference wrapper code, not when using the method of appending 3D info directly to sound buffers.

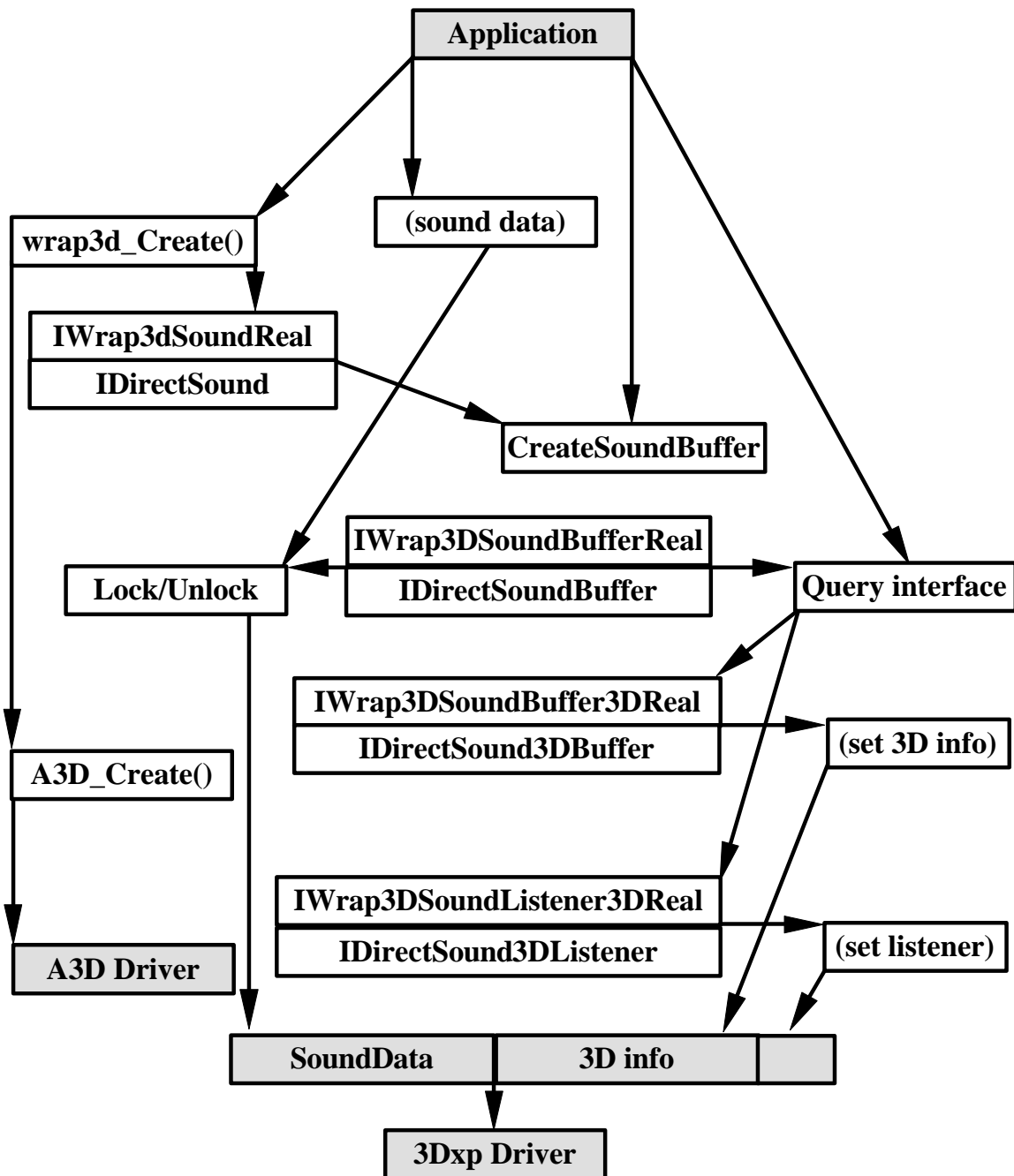
The 3Dxp extension layer:

- Adds a simple layer to DirectSound 3D and to the Game/Application

3Dxp Data Flow



3Dxp Reference Code Path:



Note: The above flow diagram represents the 3Dxp Reference Code (described in a document found on DiamondWare's Web site at <http://dw.com/3dpxp>). The 3Dxp Reference Code is provided as C++ source code, which is included in the 3Dxp package. The above flow diagram depicts the common reference code and two different driver implementations. One driver method is based on the 3Dxp protocol and has driver interface hardware independent code included in this guideline. The second is based on Aureal's method and works only on A3D licensed hardware.

Game/Application Guidelines

This document assumes that the reader is knowledgeable about DirectSound, and is at least familiar with 3D audio concepts, as discussed in the 3D Audio Primer paper which is an companion document to the 3Dxp Guideline.

The Theory:

DirectSound streams are each kept in their own buffers. They are allocated by DirectSound at the application's request, and the buffer size is specified by the application. 3Dxp-aware applications can allocate a buffer slightly larger (16+132 bytes) than the audio data, storing the 3Dxp-specific information at the end. This information includes all the DirectSound parameters unmodified.

To determine if the audio device driver is 3Dxp-compliant, the application should allocate a special-purpose DirectSound buffer, at least large enough to contain the 3Dxp query struct. The application puts a magic number in the buffer, and then reads the data struct returned by the driver. This struct will describe driver capabilities and remaining resources.

The Details:

Declarations for 3Dxp:

```
#define 3dxp_QUERY      "DEV3DCompliant?"
#define 3dxp_SIGNATURE "!t nailpmoCD3VED"
#define 3dxp_LOCALIZE  "DEV3D Localize!"

typedef struct
{
    char    szMagicString[16];
    DWORD  dw3Dxpversion;    //hi word is major, lo word is minor
    DWORD  dwMaxHighQuality3D;
    DWORD  dwMaxMediumQuality3D;
    DWORD  dwMaxLowQuality3D;
    DWORD  dwFreeHighQuality3D;
    DWORD  dwFreeMediumQuality3D;
    DWORD  dwFreeLowQuality3D;
    DWORD  dwReserved[5];
} 3dxp_INFO;
```

Querying a 3Dxp Driver:

To detect a 3Dxp-compliant driver, the application must:

- Create a DirectSound buffer to contain the 3dxp_INFO struct (at least 64 bytes)
- Call IDirectSoundBuffer::Lock()
- Write a 3dxp_QUERY string to the first 16 bytes of the buffer
- Call IDirectSoundBuffer::Unlock()
- Call IDirectSoundBuffer::Lock()

If the first 16 bytes match the 3Dxp_SIGNATURE string, the driver is 3Dxp-compliant (magic string)

The minor version number will be used to differentiate internal details and insignificant specification changes. The major version number will change if new enhancements to this interface are provided. All such changes will be backwardly compatible to the specification described in this document. Currently there is only one version which is supported as part of this guideline.

The next 6 fields require some explanation. Many hardware systems will be able to localize a limited number of sounds at full quality, degrading to medium for several more voices, and then finally to low for more voices. Beyond that, the hardware cannot localize sounds at all. So, this structure tells you the maximum and current totals for each of these 3 quality levels.

The application may make this query at any time. The same DirectSound buffer may be used, although the program would have to rewrite the 3dxp_QUERY string into it each time.

The application should create a separate IDirectSoundBuffer object for this query which will not be used for audio. Unlike with localization, the query operation is not really compatible with audio playback.

Localizing a Sound:

To localize a sound, the application must:

- Allocate a DirectSound buffer 3dxp_EXTRABUF 132 bytes larger than the sound data
- Call IDirectSoundBuffer::Lock()
- Write a 3dxp_LOCALIZE string into extra buffer region (end of buffer)
- Write a DWORD priority (higher numbers are higher priorities)
- Write a filled-in DS3DLISTENER struct
- Write a filled-in DS3DBUFFER struct
- Call IDirectSoundBuffer::Unlock()

The sound data itself is stored at the beginning of the buffer. The last 3dxp_EXTRABUF bytes are used for communicating the out-of-band localization information.

3dxp_EXTRABUF is declared in 3DXP.H; it's the total size needed to store the magic string, priority, DS3DLISTENER, and DS3DBUFFER structs.

If there are more 3D sounds than hardware capacity to localize them, the priority parameter helps the driver determine which voices get high-quality, medium-quality, low-quality, or no localization.

Since there is only one listener, each change to its parameters affects all sound streams.

The application can reposition the sound at any time, whether or not it also updates the sound data. The 3Dxp driver scrutinizes all buffer-lock calls which include the last 3dxp_EXTRABUF bytes.

The IDirectSoundBuffer::Lock() call writes the current values of all parameters into the memory region returned to the application. Besides providing a parameter query mechanism, any parameters not changed by the application will retain their values.

A value of 0xFFFFFFFF also indicates that a field should remain unchanged. This technique is useful for the first time a sound is written. Otherwise, without knowing the default values, the application would have to write the 3dxp_LOCALIZE string, call Unlock() and then Lock() again.

Reference Code:

The 3Dxp Reference Code was written to show by example how to use 3Dxp in a real-world program. The Code implements an intelligent wrapper for Microsoft's DirectSound. It uses 3Dxp or A3D for 3D localization processing (if available) or otherwise Microsoft's DirectSound3D.

The wrapper provides an API which is virtually identical to DirectSound. Here's how to use the Reference Code in programming application.

At a Minimum:

To use WRAP3D, you must replace all calls to DirectSoundCreate() with wrap3d_Create(). You should insert the line:

```
#include "wrap3d.h"
```

into any source file which makes this function call. Add WRAP3D.CPP to your project. Finally, add DSOUND.LIB and DXGUID.LIB (both provided with DirectX) to the library list in the makefile.

That's it. Your program will now exploit 3Dxp or A3D for 3D audio, if compliant hardware/software is available.

How It Works (briefly):

Internally, wrap3d.cpp implements C++ classes for each DirectSound object. These are regular C++ classes, not COM objects. They are allocated with the documented DirectSound function call, and deallocated via a call to their Release() methods, however, exactly like DirectSound objects. They are also available to C, through the same mechanism used by DirectSound: a struct with a member called lpVtbl which contains a function pointer for each method.

Errata/App Notes:

Bugs:

This code has been thoroughly reviewed and tested. However, like all other software, it may contain bugs. If you should find one, please email keith@dw.com describing the problem. Or, better yet fix the bug, send a note and a description of both the error and the changes you made to fix it. Fixes will be made in a subsequent release and you will receive credit at the top of the C++ source file.

C vs. C++

This code will work with C or C++, just like DirectSound itself. Being that it's implemented in C++, it uses operator new and operator delete. It has been tested in a C DirectSound program, with no problems. This author is not aware of any difficulties of mixing new/delete with malloc/free (as long as they're used on different memory blocks), even in C programs.

Potential Problems:

IDirectSoundBuffer does not have 3D-localization calls. It provides a QueryInterface() call which returns a pointer to a separate object of type IDirectSound3DBuffer. This object provides the 3D calls. Although they may appear to be separate, they are surely attached internally to DirectSound. Calling Release() on the IDirectSoundBuffer object and then trying to use the corresponding IDirectSound3DBuffer object afterwards could cause unexpected behaviors.

Driver Guidelines

Only two commands are used - Lock() and Unlock().

3Dxp Driver Sample Code:

```

HRESULT __stdcall   IDsDriverBuffer_Lock(
    IN PIDS DRIVERBUFFER pIDsDriverBuffer,
    OUT LPLPVOID ppvAudio1,
    OUT LPDWORD pdwLen1,
    OUT LPLPVOID ppvAudio2,
    OUT LPDWORD pdwLen2,
    IN DWORD dwWritePosition,
    IN DWORD dwWriteLen,
    IN DWORD dwFlags)
{
    PDS DRIVERBUFFER pBufferObjContext
    =(PDS DRIVERBUFFER)pIDsDriverBuffer;
    DWORD *ptr;

    _Debug_Printf_Service( "IDsDriverBuffer_Lock" ) ;

    if((dwWritePosition + dwWriteLen) <=
        pBufferObjContext->dsd_BufferSize)
    {
        *ppvAudio1 = (DWORD *) (pBufferObjContext->dsd_LinBuffAddr
+
                                dwWritePosition);
        *pdwLen1   = dwWriteLen;
        *ppvAudio2 = NULL;
        *pdwLen2   = 0;
    }
    else
    {
        DWORD firstpart = pBufferObjContext->dsd_BufferSize -
                                dwWritePosition;

        *ppvAudio1 = (DWORD *) (pBufferObjContext->dsd_LinBuffAddr
+
                                dwWritePosition);
        *pdwLen1   = firstpart;
        *ppvAudio2 = (DWORD *) (pBufferObjContext->
>dsd_LinBuffAddr);
        *pdwLen2   = dwWriteLen - firstpart;
    }

    // Copy the latest contents of current Buffer, and the
    Listener,
    // If the buffer is a 3Dxp buffer, and the region of 3D is
    what
    // is being locked.
    if(

```

3Dxp — DirectSound 3.0 Extension API — Version 1.0

```

    (pBufferObjContext->dsd_IsIt3D ==
3Dxp_DIRECTSOUND3DBUFFER)
        &&
        (
        (
            (DWORD)*ppvAudio1 + *pdwLen1 >=
pBufferObjContext->dsd_Lin3DBuffAddr
        ) ||

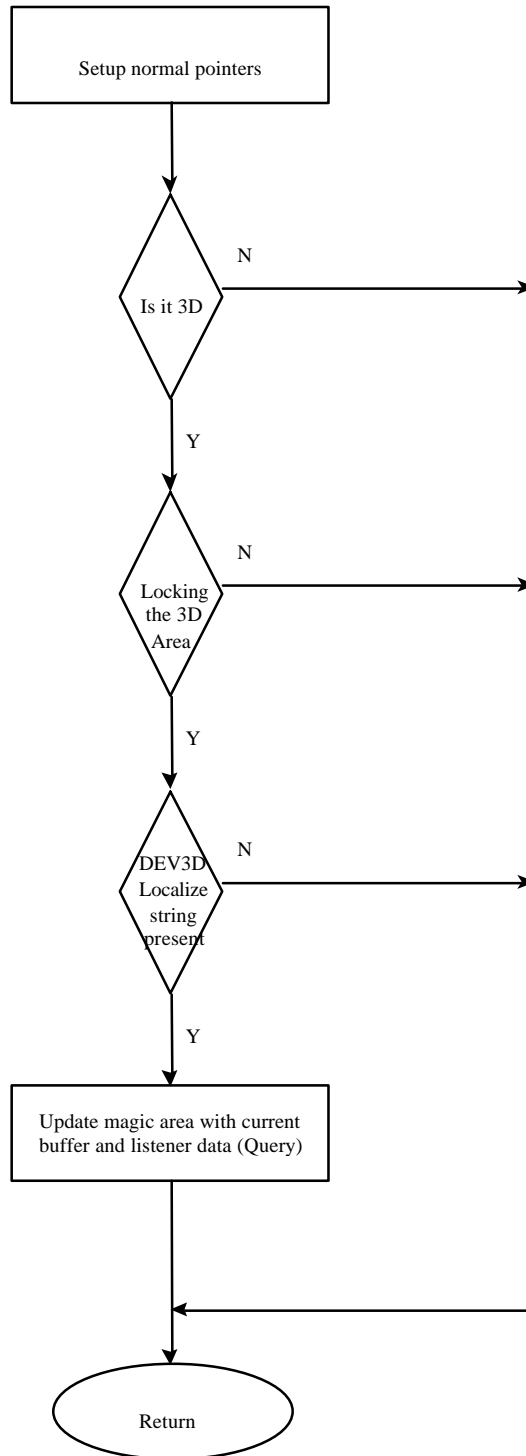
        ( ((DWORD)*ppvAudio2 != 0) &&
            ((DWORD)*ppvAudio2 + *pdwLen2 >=
pBufferObjContext->dsd_Lin3DBuffAddr)
        )
        ) &&

        (strcmp((const char *)pBufferObjContext-
>dsd_Lin3DBuffAddr,
                3DxpLOCALIZE) == 0)
        )
    {
        ptr = (DWORD *) (pBufferObjContext->dsd_Lin3DBuffAddr +
16);
        memcpy(ptr, pBufferObjContext->dsd_p3DxpContext,
                sizeof(3DxpBUFFER));
        ptr += sizeof(3DxpBUFFER) >> 2; // should be 17 DWORDS
        memcpy(ptr, &gListener, sizeof(DS3DLISTENER));
    }

    return DS_OK;
}

```

3Dxp Driver Lock Sequence:



3Dxp Driver Unlock Sequence:

