Creative OpenAL Programmer's Reference

Version 1.0

Table of Contents

Trademarks and Service Marks
Creative, Sound Blaster, and the Creative logo are registered trademarks, and Environmental Audio, EAX, and the Environmental Audio Extensions logo are trademarks of Creative Technology Ltd. in the United States and/or other countries.

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

Acknowledgments
Documentation written by Garin Hiebert. Additional input by Keith Charley, Jean-Marc Jot, Daniel Peacock, Jean-Michel Trivi, and Carlo Vogelsang.

About this Document

Introduction

OpenAL is a cross-platform three-dimensional audio API. The API's primary purpose is to allow a programmer to position audio sources in a three-dimensional space around a listener, producing reasonable fading and panning for each source so that the environment seems three-dimensional. Additional effects such as Doppler shift are also available for use. Through extensions, Creative Labs has also enhanced OpenAL with EAX and AC3 capabilities. Version 1.0 of OpenAL is appropriate for many audio applications, but was designed to be most appropriate for gaming audio.

Intended Audience

This reference guide is most appropriate for a programmer. Experience with C or C++ is not required to learn the concepts in OpenAL, but will make understanding the OpenAL source as well as sample code easier. Since there are several sample applications included with the OpenAL SDKs as well as with the source distribution, it is recommended that interested programmers take advantage of those resources.

Other OpenAL Resources

The two most important resources for additional information on OpenAL are the websites at *www.openal.org* and *http://developer.creative.com*. The main OpenAL site hosts the specification, the open source implementations, and sample code. The Creative developer's site has a section dedicated to OpenAL with SDKs showing how to use OpenAL as well as two extensions to OpenAL – EAX and AC3.

Introduction to OpenAL

Use of OpenAL revolves around the use of three fundamental objects – Buffers, Sources, and a Listener. A buffer can be filled with audio data, and can then be attached to a source. The source can then be positioned and played. How the source is heard is determined by its position and orientation relative to the Listener object (there is only one Listener). Creating a number of sources and buffers and a single listener and then updating the positions and orientations of the sources and listener dynamically can present a convincing 3D audio world.

Initializing/Exiting

The easiest way to begin initializing OpenAL is to make a call to *alutInit*. Use the parameters (0, NULL) to use the default device. Initializing in this way avoids any need to make OpenAL context calls.

If an extension such as EAX is desired, the next step is to detect the available extensions. Use *alIsExtensionPresent* to query for extensions by name, and then assign values to function pointers using *alGetProcAddress*.

To generate a set of buffers for use, use *alGetError* to reset the error state, call *alGenBuffers* to generate the number of buffers desired, and then use *alGetError* again to detect if an error was generated.

Fill the buffers with PCM data using *alBufferData*. If the PCM data is stored in WAV format on disk or in memory, *alutLoadWAVFile* or *alutLoadWAVMem* can be used to retrieve it and provide the information needed by *alBufferData*.

To generate a set of sources for use, use *alGetError* to reset the error state, call *alGenSources* to generate the number of sources desired, and then use *alGetError* again to detect if an error was generated.

Buffers are attached to sources using *alSourcei*.

Before exiting the program, call *alutExit* to clean up OpenAL.

Example:

```
alutInit(0, NULL);  // Initialize OpenAL

alGetError();  // Clear Error Code

// Check for EAX 2.0 support
g_bEAX = alIsExtensionPresent((ALubyte*)"EAX2.0");
if (g_bEAX == AL_TRUE)
{
        sprintf((char*)szFnName, "EAXSet");
        eaxSet = (EAXSet)alGetProcAddress(szFnName);
        if (eaxSet == NULL) g_bEAX = AL_FALSE;
}
if (g_bEAX == AL_TRUE)
{
        sprintf((char*)szFnName,"EAXGet");
        eaxGet = (EAXGet)alGetProcAddress(szFnName);
        if (eaxGet == NULL) g_bEAX = AL_FALSE;
}

// Generate Buffers
alGenBuffers(NUM_BUFFERS, g_Buffers);
if ((error = alGetError()) != AL_NO_ERROR)
{
        DisplayALError("alGenBuffers :", error);
```

```
        exit(-1);
}

// Load test.wav
alutLoadWAVFile("test.wav",&format,&data,&size,&freq,&loop);
if ((error = alGetError()) != AL_NO_ERROR)
{
        DisplayALError("alutLoadWAVFile test.wav : ", error);
        // Delete Buffers
        alDeleteBuffers(NUM_BUFFERS, g_Buffers);
        exit(-1);
}

// Copy test.wav data into AL Buffer 0
alBufferData(g_Buffers[0],format,data,size,freq);
if ((error = alGetError()) != AL_NO_ERROR)
{
        DisplayALError("alBufferData buffer 0 : ", error);
        // Delete buffers
        alDeleteBuffers(NUM_BUFFERS, g_Buffers);
        exit(-1);
}

// Unload test.wav
alutUnloadWAV(format,data,size,freq);
if ((error = alGetError()) != AL_NO_ERROR)
{
        DisplayALError("alutUnloadWAV : ", error);
        // Delete buffers
        alDeleteBuffers(NUM_BUFFERS, g_Buffers);
        exit(-1);
}


// Generate Sources
alGenSources(1,source);
if ((error = alGetError()) != AL_NO_ERROR)
{
        DisplayALError("alGenSources 1 : ", error);
        return;
}

// Attach buffer 0 to source
alSourcei(source[0], AL_BUFFER, g_Buffers[0]);
if ((error = alGetError()) != AL_NO_ERROR)
        DisplayALError("alSourcei AL_BUFFER 0 : ", error);
```

Listener Properties

For every context, there is automatically one Listener object. The *alListener[f, 3f, fv, i]* and *alGetListener[f, 3f, fv, i]* functions can be used to set or retrieve the following listener properties:

Property	Data Type	Description
AL_GAIN	f	"master gain" value should be positive
AL_POSITION	3f, fv	X, Y, Z position
AL_VELOCITY	3f, fv	velocity vector
AL_ORIENTATION	fv	orientation expressed as "at" and "up" vectors

Example:

```
ALfloat listenerPos[]={0.0,0.0,0.0};
ALfloat listenerVel[]={0.0,0.0,0.0};
ALfloat listenerOri[]={0.0,0.0,-1.0, 0.0,1.0,0.0}; // "at", then "up"
```

```
// Set Listener attributes

// Position ...
alListenerfv(AL_POSITION,listenerPos);
if ((error = alGetError()) != AL_NO_ERROR)
{
	DisplayALError("alListenerfv POSITION : ", error);
	exit(-1);
}

// Velocity ...
alListenerfv(AL_VELOCITY,listenerVel);
if ((error = alGetError()) != AL_NO_ERROR)
{
	DisplayALError("alListenerfv VELOCITY : ", error);
	exit(-1);
}

// Orientation ...
alListenerfv(AL_ORIENTATION,listenerOri);
if ((error = alGetError()) != AL_NO_ERROR)
{
	DisplayALError("alListenerfv ORIENTATION : ", error);
	exit(-1);
}
```

Buffer Properties

Each buffer generated by *alGenBuffers* has properties which can be retrieved. The *alGetBuffer[f, i]* function can be used to retrieve the following buffer properties:

Property	Data Type	Description
AL_ FREQUENCY	i	frequency of buffer in Hz
AL_ BITS	i	bit depth of buffer
AL_ CHANNELS	i	number of channels in buffer > 1 is valid, but buffer won't be positioned when played
AL_ SIZE	i	size of buffer in bytes
AL_DATA	i	original location where data was copied from generally useless, as was probably freed after buffer creation

Example:

```
// Retrieve Buffer Frequency
alBufferi(g_Buffers[0], AL_FREQUENCY, iFreq);
```

Source Properties

Each source generated by *alGenSources* has properties which can be set or retrieved. The *alSource[f, 3f, fv, i]* and *alGetSource[f, 3f, fv, i]* functions can be used to set or retrieve the following source properties:

Property	Data Type	Description
AL_PITCH	f	pitch multiplier always positive
AL_GAIN	f	source gain value should be positive
AL_MAX_DISTANCE	f	used with the Inverse Clamped Distance Model to set the distance where there will no longer be any attenuation of the source
AL_ROLLOFF_FACTOR	f	the rolloff rate for the source default is 1.0
AL_REFERENCE_DISTANCE	f	the distance under which the volume for the source would normally drop by half (before being influenced by rolloff factor or AL_MAX_DISTANCE)
AL_MIN_GAIN	f	the minimum gain for this source
AL_MAX_GAIN	f	the maximum gain for this source
AL_CONE_OUTER_GAIN	f	the gain when outside the oriented cone
AL_CONE_INNER_ANGLE	f, i	the gain when inside the oriented cone
AL_CONE_OUTER_ANGLE	f, i	outer angle of the sound cone, in degrees default is 360
AL_POSITION	fv, 3f	X, Y, Z position
AL_VELOCITY	fv, 3f	velocity vector
AL_DIRECTION	fv, 3f	direction vector
AL_SOURCE_RELATIVE	i	determines if the positions are relative to the listener default is AL_FALSE
AL_LOOPING	i	turns looping on (AL_TRUE) or off (AL_FALSE)
AL_BUFFER	i	the ID of the attached buffer
AL_SOURCE_STATE	i	the state of the source (AL_STOPPED, AL_PLAYING, …)
AL_BUFFERS_QUEUED*	i	the number of buffers queued on this source
AL_BUFFERS_PROCESSED*	i	the number of buffers in the queue that have been processed

** Read Only (alGetSourcei)*

Example:

```
alGetError(); // clear error state
alSourcef(source[0],AL_PITCH,1.0f);
if ((error = alGetError()) != AL_NO_ERROR)
        DisplayALError("alSourcef 0 AL_PITCH : \n", error);

alSourcef(source[0],AL_GAIN,1.0f);
if ((error = alGetError()) != AL_NO_ERROR)
        DisplayALError("alSourcef 0 AL_GAIN : \n", error);

alSourcefv(source[0],AL_POSITION,source0Pos);
if ((error = alGetError()) != AL_NO_ERROR)
        DisplayALError("alSourcefv 0 AL_POSITION : \n", error);

alSourcefv(source[0],AL_VELOCITY,source0Vel);
if ((error = alGetError()) != AL_NO_ERROR)
        DisplayALError("alSourcefv 0 AL_VELOCITY : \n", error);

alSourcei(source[0],AL_LOOPING,AL_FALSE);
if ((error = alGetError()) != AL_NO_ERROR)
        DisplayALError("alSourcei 0 AL_LOOPING true: \n", error);
```

Queuing Buffers on a Source

To continuously stream audio from a source without interruption, buffer queuing is required. To use buffer queuing, the buffers and sources are generated in the normal way, but *alSourcei* is not used to attach the buffers to the source. Instead, the functions *alSourceQueueBuffers* and *alSourceUnqueueBuffers* are used. The program can attach a buffer or a set of buffers to a source using alSourceQueueBuffers, and then call alSourcePlay on that source. While the source is playing, *alSourceUnqueueBuffers* can be called to remove buffers which have already played. Those buffers can then be filled with new data or discarded. New or refilled buffers can then be attached to the playing source using *alSourceQueueBuffers*. As long as there is always a new buffer to play in the queue, the source will continue to play.

Although some implementations of OpenAL may not enforce the following restrictions on queuing, it is recommended to observe the following additional rules:

1) A source that will be used for streaming should not have its first buffer attached using *alSourcei* – always use *alSourceQueueBuffers* to attach buffers to streaming sources.

2) All buffers attached to a source using *alSourceQueueBuffers* should have the same audio format.

Example:

```
// attach first set of buffers using queuing mechanism
alSourceQueueBuffers(Sources[0], NUMBUFFERS, Buffers);
if ((error = alGetError()) != AL_NO_ERROR)
       DisplayALError("alSourceQueueBuffers : ", error);
// turn off looping
alSourcei(Sources[0], AL_LOOPING, AL_FALSE);
// Start playing source
alSourcePlay(Sources[0]);
if ((error = alGetError()) != AL_NO_ERROR)
       DisplayALError("alSourcePlay source 0 : ", error);

ALuint count = 0;
ALuint buffersreturned = 0;
ALboolean bFinishedPlaying = AL_FALSE;
ALuint buffersinqueue = NUMBUFFERS;

while (!bFinishedPlaying)
{
  // Get status
  alGetSourceiv(Sources[0], AL_BUFFERS_PROCESSED, &processed);
  // If some buffers have been played, unqueue them
  // then load new audio into them, then add them to the queue
  if (processed > 0)
  {
    buffersreturned += processed;

    // Pseudo code for Streaming with Open AL
    // while (processed)
    //         Unqueue a buffer
    //         Load audio data into buffer
    //               (returned by UnQueueBuffers)
    //         if successful
    //                 Queue buffer
    //                 processed--
    //         else
    //                 buffersinqueue--
    //                 if buffersinqueue == 0
    //                         finished playing !

    while (processed)
    {
```

```
            alSourceUnqueueBuffers(Sources[0], 1, &BufferID);
            if ((error = alGetError()) != AL_NO_ERROR)
            {
                  DisplayALError("alSourceUnqueueBuffers 1 : ", error);
            }
            if (!bFinished)
            {
              DataToRead = (DataSize > BSIZE) ? BSIZE : DataSize;
              if (DataToRead == DataSize) bFinished = AL_TRUE;
              fread(data, 1, DataToRead, fp);
              DataSize -= DataToRead;
              if (bFinished == AL_TRUE)
              {
                    memset(data + DataToRead, 0, BSIZE - DataToRead);
              }
              alBufferData(BufferID, Format, data, DataToRead, wave.SamplesPerSec);
              if ((error = alGetError()) != AL_NO_ERROR)
                    DisplayALError("alBufferData : ", error);
              // Queue buffer
              alSourceQueueBuffers(Sources[0], 1, &BufferID);
              if ((error = alGetError()) != AL_NO_ERROR)
                    DisplayALError("alSourceQueueBuffers 1 : ", error);
              processed--;
            } else
            {
              buffersinqueue--;
              processed--;
              if (buffersinqueue == 0)
              {
                    bFinishedPlaying = true;
                    break;
              }
            }
          }
      }
```

Doppler Shift

If velocities are applied to the Listener object or to any Source object, then Doppler shift will be applied to the audio. In Creative implementations of OpenAL, the following formula is used to calculate Doppler shift:

DV = AL_DOPPLER_VELOCITY
DF = AL_DOPPLER_FACTOR
vl = listener velocity (scalar value along source-to-listener vector)
vs = source velocity (scalar value along source-to-listener vector)
f = frequency of sample
f' = Doppler shifted frequency

f' = f * (DV – DF * vl) / (DV + DF * vs)

The Doppler Velocity represents the speed of sound. The default Doppler Velocity is 343. If units other than meters/second are being used, then the Doppler velocity should be changed accordingly.

The Doppler factor is used to exaggerate or de-emphasize the Doppler shift.

Example:

```
alGetError(); // clear error state
alDopplerVelocity(1132); // using feet/second - change propagation velocity
alDopplerFactor(1.2); // exaggerate pitch shift by 20%
if ((error = alGetError()) != AL_NO_ERROR) DisplayALError("alDopplerX : ", error);
```

Error Handling

The error state of OpenAL can be retrieved at any time using *alGetError*. *alGetError* clears the error state of OpenAL when it is called, so it is common for an OpenAL application to call *alGetError* at the beginning of a critical operation to clear the error state, perform the critical operation, and then use *alGetError* again to test whether or not an error occurred.

Error Codes:

Error Code	Description
AL_NO_ERROR	there is not currently an error
AL_INVALID_NAME	a bad name (ID) was passed to an OpenAL function
AL_INVALID_ENUM	an invalid enum value was passed to an OpenAL function
AL_INVALID_VALUE	an invalid value was passed to an OpenAL function
AL_INVALID_OPERATION	the requested operation is not valid
AL_OUT_OF_MEMORY	the requested operation resulted in OpenAL running out of memory

Example:

```
alGetError();  // Clear Error Code

// Generate Buffers
alGenBuffers(NUM_BUFFERS, g_Buffers);
if ((error = alGetError()) != AL_NO_ERROR)
{
        DisplayALError("alGenBuffers :", error);
        exit(-1);
}
```

Core OpenAL Functions

Buffer-Related

alGenBuffers

Description:

This function generates one or more buffers.

C Specification:
ALvoid alGenBuffers(ALsizei n,ALuint *buffers);

Parameters:

n	The number of buffers to be generated
**buffers*	Pointer to an array of ALuint values which will store the names of the new buffers

Return Value:

None

Remarks:

If the requested number of buffers cannot be created, an error will be generated which can be detected with alGetError. If an error occurs, no buffers will be generated. If n equals zero, alGenBuffers does nothing and does not return an error.

alDeleteBuffers

Description:

This function deletes one or more buffers.

C Specification:
ALvoid alDeleteBuffers(ALsizei n,ALuint *buffers);

Parameters:

n	The number of buffers to be deleted
**buffers*	Pointer to an array of buffer names identifying the buffers to be deleted

Return Value:

None

Remarks:

If the requested number of buffers cannot be deleted, an error will be generated which can be detected with alGetError. If an error occurs, no buffers will be deleted. If n equals zero, alDeleteBuffers does nothing and will not return an error.

alIsBuffer

Description:

This function tests if a buffer name is valid.

C Specification:
Alboolean alIsBuffer(ALuint buffer);

Parameters:

buffer A buffer name to be tested for validity

Return Value:

Boolean value AL_TRUE if the buffer name is valid or AL_FALSE if the buffer name is not valid.

Remarks:

None

alBufferData

Description:

This function fills a buffer with audio data.

C Specification:
ALvoid alBufferData(ALuint buffer,ALenum format,ALvoid *data,ALsizei size,ALsizei freq);

Parameters:

buffer Buffer name to be filled with data

format Format type from among the following:
AL_FORMAT_MONO8
AL_FORMAT_MONO16
AL_FORMAT_STEREO8
AL_FORMAT_STEREO16

**data* Pointer to the audio data

size The size of the audio data in bytes

freq The frequency of the audio data

Return Value:

None

Remarks:

None

alGetBufferf

Description:

This function retrieves a floating point property of a buffer.

C Specification:
ALvoid alGetBufferf(ALuint buffer,ALenum pname,ALfloat *value);

Parameters:

buffer Buffer name whose attribute is being retrieved

pname The name of the attribute to be retrieved

**value* A pointer to an ALfloat to hold the retrieved data

Return Value:

None

Remarks:

There are no ALfloat attributes for buffers at this time.

alGetBufferi

Description:

This function retrieves an integer property of a buffer.

C Specification:
ALvoid alGetBufferi(ALuint buffer,ALenum pname,ALint *value);

Parameters:

buffer Buffer name whose attribute is being retrieved

pname The name of the attribute to be retrieved:
AL_FREQUENCY
AL_BITS
AL_CHANNELS

AL_SIZE
AL_DATA

**value* A pointer to an ALint to hold the retrieved data

Return Value:

None

Remarks:

None

Source-Related

alGenSources

Description:

This function generates one or more sources.

C Specification:
ALvoid alGenSources(ALsizei n,ALuint *sources);

Parameters:

n The number of sources to be generated

**sources* Pointer to an array of ALuint values which will store the names of the new sources

Return Value:

None

Remarks:

If the requested number of sources cannot be created, an error will be generated which can be detected with alGetError. If an error occurs, no sources will be generated. If n equals zero, alGenSources does nothing and does not return an error.

alDeleteSources

Description:

This function deletes one or more sources.

C Specification:
ALvoid alDeleteSources(ALsizei n,ALuint *sources);

Parameters:

n The number of sources to be deleted

**sources* Pointer to an array of source names identifying the sources to be deleted

Return Value:

None

Remarks:

If the requested number of sources cannot be deleted, an error will be generated which can be detected with alGetError. If an error occurs, no sources will be deleted. If n equals zero, alDeleteSources does nothing and will not return an error.

alIsSource

Description:

This function tests if a source name is valid.

C Specification:
Alboolean alIsSource(ALuint source);

Parameters:

source A source name to be tested for validity

Return Value:

Boolean value AL_TRUE if the source name is valid or AL_FALSE if the source name is not valid.

Remarks:

None

alSourcef

Description:

This function sets a floating point property of a source.

C Specification:
ALvoid alSourcef(ALuint source,ALenum pname,ALfloat value);

Parameters:

source Source name whose attribute is being set

pname The name of the attribute to set:
AL_PITCH
AL_GAIN
AL_MAX_DISTANCE

AL_ROLLOFF_FACTOR
AL_REFERENCE_DISTANCE
AL_MIN_GAIN
AL_MAX_GAIN
AL_CONE_OUTER_GAIN

value The value to set the attribute to

Return Value:

None

Remarks:

None

alSourcefv

Description:

This function sets a floating point-vector property of a source.

C Specification:
ALvoid alSourcefv(ALuint source,ALenum pname,ALfloat *values);

Parameters:

source Source name whose attribute is being set

pname The name of the attribute being set:
AL_POSITION
AL_VELOCITY
AL_DIRECTION

**values* A pointer to the vector to set the attribute to

Return Value:

None

Remarks:

This function is an alternative to alSource3f.

alSource3f

Description:

This function sets a source property requiring three floating point values.

C Specification:

ALvoid alSource3f(ALuint source,ALenum pname,ALfloat v1,ALfloat v2,ALfloat v3);

Parameters:

source Source name whose attribute is being set

pname The name of the attribute to set:
AL_POSITION
AL_VELOCITY
AL_DIRECTION

v1, v2, v3 The three ALfloat values which the attribute will be set to

Return Value:

None

Remarks:

This function is an alternative to alSourcefv.

alSourcei

Description:

This function sets an integer property of a source.

C Specification:
ALvoid alSourcei(ALuint source,ALenum pname,ALint value);

Parameters:

source Source name whose attribute is being set

pname The name of the attribute to set:
AL_SOURCE_RELATIVE
AL_CONE_INNER_ANGLE
AL_CONE_OUTER_ANGLE
AL_LOOPING
AL_BUFFER
AL_SOURCE_STATE

value The value to set the attribute to

Return Value:

None

Remarks:

None

alGetSourcef

Description:

This function retrieves a floating point property of a source.

C Specification:
ALvoid alGetSourcef(ALuint source,ALenum pname,ALfloat *value);

Parameters:

source Source name whose attribute is being retrieved

pname The name of the attribute to retrieve:
AL_PITCH
AL_GAIN
AL_MIN_GAIN
AL_MAX_GAIN
AL_MAX_DISTANCE
AL_ROLLOFF_FACTOR
AL_CONE_OUTER_GAIN
AL_CONE_INNER_ANGLE
AL_CONE_OUTER_ANGLE
AL_REFERENCE_DISTANCE

**value* A pointer to the floating point value being retrieved

Return Value:

None

Remarks:

None

alGetSourcefv

Description:

This function retrieves a floating point-vector property of a source.

C Specification:
ALvoid alGetSourcefv(ALuint source,ALenum pname,ALfloat *values);

Parameters:

source Source name whose attribute is being retrieved

pname The name of the attribute being retrieved:
AL_POSITION
AL_VELOCITY
AL_DIRECTION

**values* A pointer to the vector to retrieve

Return Value:

None

Remarks:

None

alGetSourcei

Description:

This function retrieves an integer property of a source.

C Specification:
ALvoid alGetSourcei(ALuint source,ALenum pname,ALint *value);

Parameters:

source Source name whose attribute is being retrieved

pname The name of the attribute to retrieve:
AL_SOURCE_RELATIVE
AL_BUFFER
AL_SOURCE_STATE
AL_BUFFERS_QUEUED
AL_BUFFERS_PROCESSED

**value* A pointer to the integer value being retrieved

Return Value:

None

Remarks:

None

alSourcePlay

Description:

This function plays a source.

C Specification:
ALvoid alSourcePlay(ALuint source);

Parameters:

source The name of the source to be played

Return Value:

None

Remarks:

The playing source will have its state changed to AL_PLAYING.

alSourcePlayv

Description:

This function plays a set of sources.

C Specification:
ALvoid alSourcePlayv(ALsizei n,ALuint *sources);

Parameters:

n The number of sources to be played

**sources* A pointer to an array of sources to be played

Return Value:

None

Remarks:

The playing sources will have their state changed to AL_PLAYING.

alSourcePause

Description:

This function pauses a source.

C Specification:
ALvoid alSourcePause(ALuint source);

Parameters:

source The name of the source to be paused

Return Value:

None

Remarks:

The paused source will have its state changed to AL_PAUSED.

alSourcePausev

Description:

This function pauses a set of sources.

C Specification:
ALvoid alSourcePausev(ALsizei n,ALuint *sources);

Parameters:

n The number of sources to be paused

**sources* A pointer to an array of sources to be paused

Return Value:

None

Remarks:

The paused sources will have their state changed to AL_PAUSED.

alSourceStop

Description:

This function stops a source.

C Specification:
ALvoid alSourceStop(ALuint source);

Parameters:

source The name of the source to be stopped

Return Value:

None

Remarks:

The stopped source will have its state changed to AL_STOPPED.

alSourceStopv

Description:

This function stops a set of sources.

C Specification:
ALvoid alSourceStopv(ALsizei n,ALuint *sources);

Parameters:

n	The number of sources to stop
**sources*	A pointer to an array of sources to be stopped

Return Value:

None

Remarks:

The stopped sources will have their state changed to AL_STOPPED.

alSourceRewind

Description:

This function stops the source and sets its state to AL_INITIAL.

C Specification:
ALvoid alSourceRewind(ALuint source);

Parameters:

source	The name of the source to be rewound

Return Value:

None

Remarks:

None

alSourceRewindv

Description:

This function stops a set of sources and sets all their states to AL_INITIAL.

C Specification:
ALvoid alSourceRewindv(ALsizei n,ALuint *sources);

Parameters:

n The number of sources to be rewound

**sources* A pointer to an array of sources to be rewound

Return Value:

None

Remarks:

None

alSourceQueueBuffers

Description:

This function queues a set of buffers on a source.

C Specification:
ALvoid alSourceQueueBuffers(ALuint source, ALsizei n, ALuint* buffers);

Parameters:

source The name of the source to queue buffers onto

n The number of buffers to be queued

**buffers* A pointer to an array of buffer names to be queued

Return Value:

None

Remarks:

None

alSourceUnqueueBuffers

Description:

This function unqueues a set of buffers attached to a source.

C Specification:
ALvoid alSourceUnqueueBuffers(ALuint source, ALsizei n, ALuint* buffers);

Parameters:

source	The name of the source to unqueue buffers from
n	The number of buffers to be unqueued
**buffers*	A pointer to an array of buffer names that were removed

Return Value:

None

Remarks:

The unqueue operation will only take place if all n buffers can be removed from the queue.

Listener-Related

alListenerf

Description:

This function sets a floating point property for the listener.

C Specification:
ALvoid alListenerf(ALenum pname,ALfloat value);

Parameters:

pname	The name of the attribute to be set
value	The ALfloat value to set the attribute to

Return Value:

None

Remarks:

None

alListener3f

Description:

This function sets a floating point property for the listener.

C Specification:
ALvoid alListener3f(ALenum pname,ALfloat v1,ALfloat v2,ALfloat v3);

Parameters:

pname The name of the attribute to set:

AL_POSITION
AL_VELOCITY

v1, v2, v3 The value to set the attribute to

Return Value:

None

Remarks:

None

alListenerfv

Description:

This function sets a floating point-vector property of the listener.

C Specification:
ALvoid alListenerfv(ALenum pname,ALfloat *values);

Parameters:

pname The name of the attribute to be set:

AL_POSITION
AL_VELOCITY
AL_ORIENTATION

**values* Pointer to floating point-vector values

Return Value:

None

Remarks:

None

alListeneri

Description:

This function sets an integer property of the listener.

C Specification:
ALvoid alListeneri(ALenum pname,ALint value);

Parameters:

pname The name of the attribute to be set

value The integer value to set the attribute to

Return Value:

None

Remarks:

There are no integer listener attributes at this time.

alGetListenerf

Description:

This function retrieves a floating point property of the listener.

C Specification:
ALvoid alGetListenerf(ALenum pname,ALfloat *value);

Parameters:

pname The name of the attribute to be retrieved

AL_GAIN

**value* A pointer to the floating point value being retrieved

Return Value:

None

Remarks:

None

alGetListener3f

Description:

This function retrieves a set of three floating point values from a property of the listener.

C Specification:
ALvoid alGetListener3f(ALenum pname,ALfloat *v1,ALfloat *v2,ALfloat *v3);

Parameters:

pname The name of the attribute to be retrieved

AL_POSITION
AL_VELOCITY

**v1, *v2, *v3* Pointers to the three floating point being retrieved

Return Value:

None

Remarks:

None

alGetListenerfv

Description:

This function retrieves a floating point-vector property of the listener.

C Specification:
ALvoid alGetListenerfv(ALenum pname,ALfloat *values);

Parameters:

pname The name of the attribute to be retrieved

AL_POSITION
AL_VELOCITY
AL_ORIENTATION

**values* A pointer to the floating point-vector value being retrieved

Return Value:

None

Remarks:

None

alGetListeneri

Description:

This function retrieves an integer property of the listener.

C Specification:
ALvoid alGetListeneri(ALenum pname,ALint *value);

Parameters:

pname The name of the attribute to be retrieved

**value* A pointer to the integer value being retrieved

Return Value:

None

Remarks:

There are no integer listener attributes at this time.

State-Related

alEnable

Description:

This function enables a feature of the OpenAL driver.

C Specification:
ALvoid alEnable(ALenum capability);

Parameters:

capability The name of a capability to enable

Return Value:

None

Remarks:

At the time of this writing, there are no features to be disabled using this function, so if it is called the error AL_INVALID_ENUM will be generated.

alDisable

Description:

This function disables a feature of the OpenAL driver.

C Specification:
ALvoid alDisable(ALenum capability);

Parameters:

capability The name of a capability to enable

Return Value:

None

Remarks:

At the time of this writing, there are no features to be disabled using this function, so if it is called the error AL_INVALID_ENUM will be generated.

alIsEnabled

Description:

This function returns a boolean indicating if a specific feature is enabled in the OpenAL driver.

C Specification:
Alboolean alIsEnabled(ALenum capability);

Parameters:

capability The name of a capability to enable

Return Value:

AL_TRUE if the capability is enabled, AL_FALSE if the capability is disabled

Remarks:

At the time of this writing, this function always returns AL_FALSE, and since there are no capabilities defined yet, the error AL_INVALID_ENUM will also be set.

alGetBoolean

Description:

This function returns a boolean OpenAL state.

C Specification:
Alboolean alGetBoolean(ALenum pname);

Parameters:

pname The state to be queried

Return Value:

The boolean state described by pname will be returned.

Remarks:

There aren't any boolean states defined at the time of this writing, so this function will always generate the error AL_INVALID_ENUM.

alGetDouble

Description:

This function returns a double precision floating point OpenAL state.

C Specification:
Aldouble alGetDouble(ALenum pname);

Parameters:

pname The state to be queried

Return Value:

The double value described by pname will be returned

Remarks:

There aren't any double precision floating point states defined at the time of this writing, so this function will always generate the error AL_INVALID_ENUM.

alGetFloat

Description:

This function returns a floating point OpenAL state.

C Specification:
ALfloat alGetFloat(ALenum pname);

Parameters:

pname The state to be queried:

AL_DOPPLER_FACTOR

AL_DOPPLER_VELOCITY

Return Value:

The floating point state described by pname will be returned.

Remarks:

None

alGetInteger

Description:

This function returns an integer OpenAL state.

C Specification:
Alint alGetInteger(ALenum pname);

Parameters:

pname The state to be queried

AL_DISTANCE_MODEL

Return Value:

The integer state described by pname will be returned.

Remarks:

None

alGetBooleanv

Description:

This function retrieves a boolean OpenAL state.

C Specification:
ALvoid alGetBooleanv(ALenum pname,ALboolean *data);

Parameters:

pname The state to be returned

**data* A pointer to the location where the state will be stored

Return Value:

None

Remarks:

There aren't any boolean states defined at the time of this writing, so this function will always generate the error AL_INVALID_ENUM.

alGetDoublev

Description:

This function retrieves a double precision floating point OpenAL state.

C Specification:
ALvoid alGetDoublev(ALenum pname,ALdouble *data);

Parameters:

pname The state to be returned

**data* A pointer to the location where the state will be stored

Return Value:

None

Remarks:

There aren't any double precision floating point states defined at the time of this writing, so this function will always generate the error AL_INVALID_ENUM.

alGetFloatv

Description:

This function retrieves a floating point OpenAL state.

C Specification:
ALvoid alGetFloatv(ALenum pname,ALfloat *data);

Parameters:

pname The state to be returned
AL_DOPPLER_FACTOR
AL_DOPPLER_VELOCITY

**data* A pointer to the location where the state will be stored

Return Value:

None

Remarks:

None

alGetIntegerv

Description:

This function retrieves an integer OpenAL state.

C Specification:
ALvoid alGetIntegerv(ALenum pname,ALint *data);

Parameters:

pname The state to be returned

AL_DISTANCE_MODEL

**data* A pointer to the location where the state will be stored

Return Value:

None

Remarks:

None

alGetString

Description:

This function retrieves an OpenAL string property.

C Specification:
ALubyte * alGetString(ALenum pname);

Parameters:

pname The property to be returned
AL_VENDOR
AL_VERSION
AL_RENDERER
AL_EXTENSIONS

Return Value:

A pointer to a null-terminated string

Remarks:

None

alDistanceModel

Description:

This function selects the OpenAL distance model.

The AL_INVERSE_DISTANCE model works according to the following formula:

G_dB = AL_GAIN – 20log10(1 + AL_ROLLOFF_FACTOR*(distance –
AL_REFERENCE_DISTANCE)/AL_REFERENCE_DISTANCE));
G_dB = min(G_dB, AL_MAX_GAIN);
G_dB = max(G_dB, AL_MIN_GAIN);

The AL_INVERSE_DISTANCE_CLAMPED model works according to the following formula:

distance = max(distance, AL_REFERENCE_DISTANCE);
distance = min(distance, AL_MAX_DISTANCE);
G_dB = AL_GAIN – 20log10(1 + AL_ROLLOFF_FACTOR*(distance –
AL_REFERENCE_DISTANCE)/AL_REFERENCE_DISTANCE));
G_dB = min(G_dB, AL_MAX_GAIN);
G_dB = max(G_dB, AL_MIN_GAIN);

The AL_NONE model works according to the following formula:

G_db = AL_GAIN;

C Specification:
ALvoid alDistanceModel(ALenum value);

Parameters:
value The distance model to be set:
AL_NONE
AL_INVERSE_DISTANCE
AL_INVERSE_DISTANCE_CLAMPED

Return Value:

None

Remarks:

The default distance model in OpenAL is AL_INVERSE_DISTANCE.

alDopplerFactor

Description:

This function selects the OpenAL Doppler factor value.

C Specification:
ALvoid alDopplerFactor(ALfloat value);

Parameters:

value The Doppler scale value to set

Return Value:

None

Remarks:

The default Doppler factor value is 1.0.

alDopplerVelocity

Description:

This function selects the OpenAL Doppler velocity value.

C Specification:
ALvoid alDopplerVelocity(ALfloat value);

Parameters:

value The Doppler velocity value to set

Return Value:

None

Remarks:

The default Doppler velocity value is 343.0.

Error-Related

alGetError

Description:

This function returns the current error state and then clears the error state.

C Specification:
ALenumalGetError(ALvoid);

Parameters:

None

Return Value:

None

Remarks:

When an OpenAL error occurs, the error state is set and will not be changed until the error state is retrieved using alGetError. Whenever alGetError is called, the error state is cleared and the last state (the current state when the call was made) is returned. To isolate error detection to a specific portion of code, alGetError should be called before the isolated section to clear the current error state.

Extension-Related

alIsExtensionPresent

Description:

This function tests if a specific extension is available for the OpenAL driver.

C Specification:
ALboolean alIsExtensionPresent(ALubyte *extName);

Parameters:

*extName A null-terminated string describing the desired extension

Return Value:

AL_TRUE if the extension is available, AL_FALSE if the extension is not available

Remarks:

None

alGetProcAddress

Description:

This function returns the address of an OpenAL extension function.

C Specification:
ALvoid * alGetProcAddress(ALubyte *funcName);

Parameters:

*funcName A null-terminated string containing the function name

Return Value:

A pointer to the desired function is returned.

Remarks:

The return value will be NULL if the function is not found.

alGetEnumValue

Description:

This function returns the enumeration value of an OpenAL enum described by a string.

C Specification:
ALenum alGetEnumValue(ALubyte *enumName);

Parameters:

*enumName A null-terminated string describing an OpenAL enum

Return Value:

The actual ALenum for the described enum is returned.

Remarks:

Returns NULL if the string doesn't describe a valid OpenAL enum.

EAX-Related

EAXGet

Description:

This function retrieves an EAX value.

C Specification:
ALenum EAXGet(const struct _GUID *propertySetID,ALuint property,ALuint source,ALvoid *value,ALuint size);

Parameters:

**propertySetID* A pointer to the property set GUID of the object being queried (a listener or a source)

property The property being queried

source The ID of the source to be queried

**value* A pointer to the value being returned

size The size of the data storage area pointed to by *value

Return Value:

An OpenAL error code indicating if there was an error in retrieving the data

Remarks:

None

EAXSet

Description:

This function sets an EAX value.

C Specification:
ALenum EAXSet(const struct _GUID *propertySetID,ALuint property,ALuint source,ALvoid *value,ALuint size);

Parameters:

**propertySetID* A pointer to the property set GUID of the object being set (a listener or a source)

property The property being set

source The ID of the source to be set

**value* A pointer to the value being returned

size The size of the data storage area pointed to by *value

Return Value:

An OpenAL error code indicating if there was an error in setting the data

Remarks:

None

ALC Functions

alcCreateContext

Description:

This function creates a context using a specified device.

C Specification:

void * alcCreateContext(ALCdevice *dev, ALint* attrlist);

Parameters:

**dev* A pointer to a device

**attrlist* A pointer to a set of attributes:

ALC_FREQUENCY
ALC_REFRESH
ALC_SYNC

Return Value:

Returns a pointer to the new context (NULL on failure)

Remarks:

None

alcMakeContextCurrent

Description:

This function makes a specified context the current context.

C Specification:
ALCenum alcMakeContextCurrent(ALvoid *alcHandle);

Parameters:

**alcHandle* Pointer to the new context

Return Value:

Returns an error code on failure

Remarks:

None

alcProcessContext

Description:

This function tells a context to begin processing.

C Specification:
void alcProcessContext(ALvoid *alcHandle);

Parameters:

**alcHandle* Pointer to the new context

Return Value:

None

Remarks:

None

alcSuspendContext

Description:

This function suspends processing on a specified context.

C Specification:
void alcSuspendContext(ALvoid *alcHandle);

Parameters:

*alcHandle A pointer to the context to be suspended

Return Value:

None

Remarks:

None

alcDestroyContext

Description:

This function destroys a context.

C Specification:
ALCenum alcDestroyContext(ALvoid *alcHandle);

Parameters:

**alcHandle* Pointer to the new context

Return Value:

Returns a context error

Remarks:

None

alcGetError

Description:

This function retrieves the current context error state.

C Specification:
ALCenum alcGetError(ALvoid);

Parameters:

None

Return Value:

The current context error state will be returned

Remarks:

None

alcGetCurrentContext

Description:

This function retrieves the current context.

C Specification:
void * alcGetCurrentContext(ALvoid);

Parameters:

None

Return Value:

Returns a pointer to the current context

Remarks:

None

alcOpenDevice

Description:

This function opens a device by name.

C Specification:

ALCdevice *alcOpenDevice(const ALubyte *tokstr);

Parameters:

*tokstr A null-terminated string describing a device

Return Value:

Returns a pointer to the opened device

Remarks:

None

alcCloseDevice

Description:

This function closes a device by name.

C Specification:
void alcCloseDevice(ALCdevice *dev);

Parameters:

*dev A pointer to an opened device

Return Value:

None

Remarks:

None

alcIsExtensionPresent

Description:

This function queries if a specified context extension is available.

C Specification:
ALboolean alcIsExtensionPresent(ALCdevice *device, ALubyte *extName);

Parameters:

*device The device to be queried for an extension

*extName A null terminated string describing the extension

Return Value:

Returns AL_TRUE if the extension is available, AL_FALSE if the extension is not available

Remarks:

None

alcGetProcAddress

Description:

This function retrieves the address of a specified context extension function.

C Specification:
ALvoid * alcGetProcAddress(ALCdevice *device, ALubyte *funcName);

Parameters:

*device The device to be queried for the function

*funcName A null terminated string describing the function

Return Value:

Returns the address of the function, or NULL if it is not found.

Remarks:

None

alcGetEnumValue

Description:

This function retrieves the enum value for a specified enumeration name.

C Specification:
ALenum alcGetEnumValue(ALCdevice *device, ALubyte *enumName);

Parameters:

*device The device to be queried

*enumName A null terminated string describing the enum value

Return Value:

Returns the enum value described by the enumName string

Remarks:

None

alcGetString

Description:

This function returns strings related to the context.

C Specification:
ALubyte * alcGetString(ALCdevice *device, ALenum token);

Parameters:

*device The device to be queried

token An attribute to be retrieved:

ALC_DEFAULT_DEVICE_SPECIFIER
ALC_DEVICE_SPECIFIER
ALC_EXTENSIONS

Return Value:

Returns a pointer to a string.

Remarks:

None

alcGetIntegerv

Description:

This function returns integers related to the context.

C Specification:
ALvoid alcGetIntegerv(ALCdevice *device, ALenum token, ALsizei size, ALint *dest);

Parameters:

*device The device to be queried

token An attribute to be retrieved:

ALC_MAJOR_VERSION
ALC_MINOR_VERSION
ALC_ATTRIBUTES_SIZE
ALC_ALL_ATTRIBUTES

size The size of the destination buffer provided

*dest A pointer to the data to be returned

Return Value:

None

Remarks:

None

ALUT Functions

alutInit

Description:

This function initializes OpenAL.

C Specification:
void alutInit(int *argc, char *argv[]);

Parameters:

*argc A pointer to an integer with the number of arguments

*argv[] A pointer to the arguments

Return Value:

None

Remarks:

This function is not guaranteed to be included with any OpenAL distribution, as it is not part of the specification of OpenAL. This function or a similar one will probably exist, however.

alutExit

Description:

This function exits OpenAL.

C Specification:
void alutExit(ALvoid);

Parameters:

None

Return Value:

None

Remarks:

This function is not guaranteed to be included with any OpenAL distribution, as it is not part of the specification of OpenAL. This function or a similar one will probably exist, however.

alutLoadWAVFile

Description:

This function loads a WAV file into memory from a file.

C Specification:
ALboolean alutLoadWAVFile(const char *fname, ALsizei *format, ALsizei *size, ALsizei *bits, ALsizei *freq, ALboolean *loop);

Parameters:

*fname	A null-terminated string with the filename
*format	A pointer to an OpenAL format specifier
*size	A pointer to the size of the data in bytes
*bits	A pointer to the bit depth of the data
*freq	A pointer to the frequency of the data
*loop	A pointer to a looping indicator for the WAV data

Return Value:

Returns AL_TRUE if there were no problems loading the file, AL_FALSE otherwise.

Remarks:

This function is not guaranteed to be included with any OpenAL distribution, as it is not part of the specification of OpenAL. This function or a similar one will probably exist, however.

alutLoadWAVMemory

Description:

This function loads a WAV file into memory from another memory location.

C Specification:
ALvoid alutLoadWAVMemory(ALbyte *memory,ALenum *format,ALvoid **data,ALsizei *size,ALsizei *freq,ALboolean *loop)

Parameters:

*memory A pointer to the memory location of the WAV data

*format A pointer to an OpenAL format specifier

*size A pointer to the size of the data in bytes

*bits A pointer to the bit depth of the data

*freq A pointer to the frequency of the data

*loop A pointer to a looping indicator for the WAV data

Return Value:

Returns AL_TRUE if there were no problems loading the file, AL_FALSE otherwise.

Remarks:

This function is not guaranteed to be included with any OpenAL distribution, as it is not part of the specification of OpenAL. This function or a similar one will probably exist, however.

alutUnloadWAV

Description:

This function unloads a WAV file from memory and is normally used after copying the data into a buffer after an alutLoad* function.

C Specification:
ALvoid alutUnloadWAV(ALenum format, ALvoid *data, ALsizei size, ALsizei freq)

Parameters:

format An OpenAL format specifier

*data A pointer to the WAV data to be unloaded from memory

size The size of the data

freq The frequency of the data

Return Value:

None

Remarks:

This function is not guaranteed to be included with any OpenAL distribution, as it is not part of the specification of OpenAL. This function or a similar one will probably exist, however.