The OpenAL Utility Toolkit (ALUT)

Contents

Release History

Discussion of the API is done via the openal-devel mailing list.

- 2005-08-14: Version 1.0.0 by Steve Baker
- 2005-09-02: Version 1.0.1 by Sven Panne
- 2005-09-10: Version 1.0.2 by Sven Panne
- 2005-09-26: Version 1.0.3 by Sven Panne
- 2005-09-28: Version 1.0.4 by Sven Panne
- 2005-10-29: Version 1.0.5 by Sven Panne
- 2005-11-19: Version 1.0.6 by Sven Panne
- 2006-04-10: Version 1.0.7 by Sven Panne
- 2006-04-11: Version 1.1.0 by Sven Panne

Introduction

This is the OpenAL Utility Toolkit (ALUT) Reference Manual.

Licensing

Some previous versions of ALUT were released under the BSD license - others under LGPL. This version will be released exclusively under LGPL.

Some History

At the time of the first writing of this document (August 2005), ALUT was a set of undocumented semi-portable functions that were mixed up in the OpenAL library distribution. The intent had always been that ALUT would be a cleanly separated library that would be portable between systems. It was hoped that it would be well suited to producing succinct demo programs and to help new developers to get started with OpenAL. It was to do this by removing the annoying details of getting an audio application started - allowing developers to learn OpenAL without distractions such as loading sound samples from disk.

In order to move from this initial implementation to a clean API that would meet the original goals of ALUT, it was necessary to break from the past and make a clean start. The original version(s) were unnumbered - so we will arbitarily label all previous versions as 0.x.x and start this cleaned up version at release 1.0.0 to reflect changed API and implementations.

Backwards Compatibility with Version 0.x.x

There are no formal guarantees of reverse compatibility with the various versions of ALUT prior to 1.0.0. Having said that, some effort has been made to at least allow these programs to continue to run if they are recompiled against ALUT 1.0.0 or later.

The old Linux implementation of OpenAL poses a special compatibility problem: ALUT 0.x.x was not a physically separate library on this platform, it was actually part of libopenal itself. This is bad for at least two reasons: It was handled differently on other platforms and much more seriously it locked together OpenAL and ALUT releases. So a deliberate decision was made to break binary compatibility in this respect and cleanly split the libraries into an OpenAL (i.e. AL and ALC) part and an ALUT one.

If you have a program which needs such an old, deprecated "combined OpenAL/ALUT" and you are not able to recompile it for some reason (e.g. it is available in binary format only), then temporarily setting the environment variable

`LD_PRELOAD` to the full path of your installed ALUT dynamic library can help. If this really works depends on the platform, but e.g. Linux, FreeBSD, NetBSD, Solaris etc. support this mechanism. On Mac OS X there is a similar environment variable called `DYLD_INSERT_LIBRARIES`, but this has not been tested yet.

Example: Using a legacy program with the new ALUT

Let's assume that your ALUT dynamic library is at the usual location `/usr/lib/libalut.so` and you have an old program called `myOldProg`, then the following commandline in Bash syntax does the trick:

```
LD_PRELOAD="/usr/lib/libalut.so" myOldProg
```

Note that setting `LD_PRELOAD` globally might not be a good idea, because in that case the new ALUT would be loaded before *every* dynamically linked executable.

OpenGL, GLUT and using what you already know

If you are already familiar with OpenGL and its utility toolkit GLUT, then you should feel very familiar with ALUT. Wherever GLUT has 'GL', ALUT has 'AL' and wherever GLUT has 'glut', ALUT has 'alut'. 'Window' is replaced with 'Context' throughout the API.

Example: 'Hello, world' in ALUT

Here is the traditional first program for any language or library, but this time it is actually *saying* 'Hello, world!' instead of printing it:

```
#include <stdlib.h>
#include <AL/alut.h>

int
main (int argc, char **argv)
{
  ALuint helloBuffer, helloSource;
  alutInit (&argc, argv);
  helloBuffer = alutCreateBufferHelloWorld ();
  alGenSources (1, &helloSource);
  alSourcei (helloSource, AL_BUFFER, helloBuffer);
  alSourcePlay (helloSource);
  alutSleep (1);
  alutExit ();
  return EXIT_SUCCESS;
}
```

Note that there error checks are missing in the program above to keep it simple.

Compiling and Linking

All ALUT programs should contain:

```
#include <AL/alut.h>
```

The ALUT header includes `<AL/al.h>` and `<AL/alc.h>` for you so you don't need to include them again - although it does not hurt to do so. ALUT reserves the "`ALUT_`" prefix for preprocessor macros, so you should never define such a macro in your own program. Furthermore, you should not rely on any macro starting with "`ALUT_`" not mentioned in this specification.

If you are using the freealut implementation of ALUT, which is available via the OpenAL homepage, you can find out the necessary compilation flags by using one of the following commands:

```
pkg-config --cflags freealut
freealut-config --cflags
```

To find out the necessary flags for linking, use one of the following commands:

```
pkg-config --libs freealut
freealut-config --libs
```

On Windows, link with `alut.dll` and `openal32.dll`.

ALUT reserves the "`alut`" prefix for globally visible functions and variables, so you should never define such a function or variable in your own program. Furthermore, you should not rely on any such function or variable not mentioned in this specification.

The ALUT API

Error Handling

ALUT's error handling and reporting is a little bit different from the one used in OpenAL and OpenGL: All functions which can fail report success/failure via a return value, where `AL_FALSE` / `AL_NONE` / `NULL` mean failure. `alutGetError` can be used to find out what exactly went wrong.

It is guaranteed that if a function fails, no data pointed to by pointer arguments has been changed.

alutGetError

Name

`alutGetError` - return and clear the current error state

Synopsis

```
ALenum alutGetError (void);
```

Description

Any ALUT routine that fails will return `AL_FALSE` / `AL_NONE` / `NULL` and set the global error state. If a subsequent error occurs while there is still an error recorded internally, the second error will simply be ignored. Calling `alutGetError` will reset the error code to `ALUT_ERROR_NO_ERROR`. Note that the error state is *not* cleared by other successful ALUT calls.

Return Value

`alutGetError` returns the contents of the global error state, which can be one of the following values:

`ALUT_ERROR_NO_ERROR`
: No ALUT error found.

`ALUT_ERROR_OUT_OF_MEMORY`
: ALUT ran out of memory.

`ALUT_ERROR_INVALID_ENUM`
: ALUT was given an invalid enumeration token.

`ALUT_ERROR_INVALID_VALUE`
: ALUT was given an invalid value.

`ALUT_ERROR_INVALID_OPERATION`
: The operation is invalid in the current ALUT state.

`ALUT_ERROR_NO_CURRENT_CONTEXT`
: There is no current AL context.

`ALUT_ERROR_AL_ERROR_ON_ENTRY`
: There was already an AL error on entry to an ALUT function.

`ALUT_ERROR_ALC_ERROR_ON_ENTRY`
: There was already an ALC error on entry to an ALUT function.

`ALUT_ERROR_OPEN_DEVICE`
: There was an error opening the ALC device.

`ALUT_ERROR_CLOSE_DEVICE`
: There was an error closing the ALC device.

`ALUT_ERROR_CREATE_CONTEXT`
: There was an error creating an ALC context.

`ALUT_ERROR_MAKE_CONTEXT_CURRENT`
: Could not change the current ALC context.

`ALUT_ERROR_DESTROY_CONTEXT`
: There was an error destroying the ALC context.

`ALUT_ERROR_GEN_BUFFERS`
: There was an error generating an AL buffer.

`ALUT_ERROR_BUFFER_DATA`
: There was an error passing buffer data to AL.

`ALUT_ERROR_IO_ERROR`
: I/O error, consult `errno` for more details.

`ALUT_ERROR_UNSUPPORTED_FILE_TYPE`
: Unsupported file type.

`ALUT_ERROR_UNSUPPORTED_FILE_SUBTYPE`
: Unsupported mode within an otherwise usable file type.

`ALUT_ERROR_CORRUPT_OR_TRUNCATED_DATA`
: The sound data was corrupt or truncated.

Errors

`alutGetError` can be called in any ALUT state and will never fail.

alutGetErrorString

Name

`alutGetErrorString` - return an error message string given an error code

Synopsis

```
const char *alutGetErrorString (ALenum error);
```

Description

`alutGetErrorString` can be used to convert an error code into a human-readable description. The precise text of these descriptions may vary from implementation to implementation and should not be relied upon by the application.

Return Value

`alutGetErrorString` returns a pointer to an immutable zero-terminated string corresponding to the given error code.

Errors

`alutGetErrorString` can be called in any ALUT state and will never fail. An unknown error code is not considered an error and a generic description will be returned in that case.

Example: Context Handling and Error Reporting

A typical ALUT program might look like this:

```
static void
reportError (void)
{
  fprintf (stderr, "ALUT error: %s\n",
           alutGetErrorString (alutGetError ()));
  exit (EXIT_FAILURE);
}

int
main (int argc, char **argv)
{
  if (!alutInit (&argc, argv))
    {
      reportError ();
    }

  /* ...play audio for a while... */

  if (!alutExit ())
    {
      reportError ();
    }

  return EXIT_SUCCESS;
}
```

Initialization / Exit

ALUT starts in an *uninitialized* state. `alutInit` and `alutInitWithoutContext` put ALUT into the *initialized* state. Those functions must only be called when the state is *uninitialized*. `alutExit` puts ALUT back from an *initialized* state to an *uninitialized* one.

The following functions must only be called in an *initialized* state and with a current context: `alutExit`, `alutCreateBufferFromFile`,

alutCreateBufferFromFileImage, alutLoadMemoryFromFile, alutLoadMemoryFromFileImage, alutGetMIMETypes, alutCreateBufferHelloWorld, alutCreateBufferWaveform. All these functions check for AL/ALC errors on entry and immediately return ALUT_ERROR_AL_ERROR_ON_ENTRY or ALUT_ERROR_ALC_ERROR_ON_ENTRY if there was one. Note that as a consequence of these checks the AL/ALC error states for the current context/device are cleared after calling any of these functions.

alutSleep can be called in every state.

The following functions never fail and can be called in any state: alutGetError, alutGetErrorString, alutGetMajorVersion, alutGetMinorVersion.

alutInit

Name

alutInit - initialize the ALUT library and create a default current context

Synopsis

```
ALboolean alutInit (int *argcp, char **argv);
```

Description

alutInit initializes the ALUT internals and creates an OpenAL context on the default device and makes it the current OpenAL context. If you want something more complex than that (e.g. running on a non-default device or opening multiple contexts on multiple devices), you can use alutInitWithoutContext instead. alutInit examines the commandline arguments passed to it and remove those it recognizes. It is acceptable to pass two NULL pointers in settings where no useful information can be obtained from argc and argv.

Return Value

alutInit returns AL_TRUE on success or AL_FALSE on failure.

Errors

ALUT_ERROR_INVALID_VALUE
: One of the arguments was NULL, but not the other one.

ALUT_ERROR_INVALID_OPERATION
: ALUT has already been initialized.

ALUT_ERROR_OPEN_DEVICE
: There was an error opening the default ALC device.

ALUT_ERROR_CREATE_CONTEXT

There was an error creating an ALC context.

`ALUT_ERROR_MAKE_CONTEXT_CURRENT`

Could not change the current ALC context.

Example: Handling command-line options

If you pass `alutInit` the argc and argv from your main program, it will examine your command-line options and use (and remove) those options that it recognises:

```
int
main (int argc, char **argv)
{
  alutInit (&argc, argv);
  ...
}
```

Precisely which (if any) command-line options are accepted and what they control is implementation and operating system dependent. Note that some implementations will use argv[0] in debug and error messages - but this is not guaranteed by the API because it is operating-system dependent. On some OS's, `alutInit` may use initial settings from other sources such as 'registry' data, '`.alutrc`' files or shell variables. Please consult the README.xxx file for your OS if you need further details.

alutInitWithoutContext

Name

`alutInitWithoutContext` - initialize the ALUT library

Synopsis

```
ALboolean alutInitWithoutContext (int *argcp, char **argv);
```

Description

`alutInitWithoutContext` initializes the ALUT internals. It does not create any OpenAL context or device, so this has to be done via the usual ALC calls. `alutInitWithoutContext` examines the commandline arguments passed to it and remove those it recognizes. It is acceptable to pass two `NULL` pointers in settings where no useful information can be obtained from argc and argv.

Return Value

`alutInitWithoutContext` returns `AL_TRUE` on success or `AL_FALSE` on failure.

Errors

`ALUT_ERROR_INVALID_VALUE`
: One of the arguments was `NULL`, but not the other one.

`ALUT_ERROR_INVALID_OPERATION`
: ALUT has already been initialized.

alutExit

Name

`alutExit` - shutdown the ALUT library

Synopsis

```
ALboolean alutExit (void);
```

Description

When the application has finished playing audio, it should shut down ALUT using `aluExit`. This closes any OpenAL device/context that ALUT may have created in `alutInit` (but not any that the application created using ALC). After calling `alutExit`, you may subsequently call `alutInit` or `alutInitWithoutContext` again. Note that under well-behaved operating systems, it should be acceptable to simply exit from your program without bothering to call `alutExit`, relying on the OS to clean up after you. However, it is dangerous to rely on this behavior if portable operation is expected.

Return Value

`alutExit` returns `AL_TRUE` on success or `AL_FALSE` on failure.

Errors

`ALUT_ERROR_INVALID_OPERATION`
: ALUT has not been initialised.

`ALUT_ERROR_NO_CURRENT_CONTEXT`
: There is no current AL context.

`ALUT_ERROR_AL_ERROR_ON_ENTRY`
: There was already an AL error on entry to `alutExit`.

`ALUT_ERROR_ALC_ERROR_ON_ENTRY`
: There was already an ALC error on entry to `alutExit`.

`ALUT_ERROR_CLOSE_DEVICE`
: There was an error closing the ALC device created by `alutInit`.

`ALUT_ERROR_MAKE_CONTEXT_CURRENT`
: Could not release the current ALC context.

`ALUT_ERROR_DESTROY_CONTEXT`

There was an error destroying the ALC context created by `alutInit`.

Sound Sample File Loading

ALUT attempts to simplify the business of getting a simple application running by providing loaders for a range of file formats. Rather than enumerate a list of formats that will likely grow with time, the loaders are generic and try to do their best either by using OpenAL extensions if possible or by converting the sound data into standard OpenAL formats.

In order to simplify initial startup and to keep test program distributions clean, ALUT provides built-in sounds, too, that do not require disk I/O because they are built into the ALUT library. These functions may be used to write compact ALUT test/example applications with no external file dependancies whatsoever. When sending short application programs to either the ALUT or OpenAL developers as a part of bug reporting, one should endeavor to use these functions instead of loading ones own sound files.

There are eight (= 4 * 2) different loaders, corresponding to the sources and destinations of the sound data. The possible sources are:

- The loaders with a `FromFile` suffix get their sound data from a named file.
- The loaders with a `FromFileImage` suffix get their data from a continuous memory region. This region can be re-used or destroyed afterwards.
- The loaders with a `HelloWorld` suffix get their fixed data internally.
- The loaders with a `Waveform` suffix get their data via internal waveform calculation.

The possible destinations are:

- The loaders with a `alutCreateBuffer` prefix create a new OpenAL buffer and put the sound data into it. If possible, OpenAL extensions are used to avoid conversions at the ALUT level and enable the use of possible hardware/driveer features for some sound formats. Therefore, these are the preferred loaders.
- The loaders with a `alutLoadMemory` prefix allocate a new memory region with `malloc` and put the sound data into it, optionally passing back more information about the sound. The sound data is guaranteed to be in one of the four standard OpenAL formats (8/16bit monon/stereo) aftwerwards. Because no OpenAL extensions can be used here, these loaders might handle fewer sound formats than the `alutCreateBuffer` ones.

alutCreateBufferFromFile

Name

`alutCreateBufferFromFile` - load a sound file into an OpenAL buffer

Synopsis

```
ALuint alutCreateBufferFromFile (const char *filename);
```

`alutCreateBufferFromFile` tries to guess the sound data format by looking at the filename and/or the file contents and loads the sound data into an OpenAL buffer.

Return Value

On success, `alutCreateBufferFromFile` returns a handle to an OpenAL buffer containing the loaded sound. It returns `AL_NONE` on failure.

Errors

`ALUT_ERROR_OUT_OF_MEMORY`
: ALUT ran out of memory.

`ALUT_ERROR_INVALID_OPERATION`
: ALUT has not been initialised.

`ALUT_ERROR_NO_CURRENT_CONTEXT`
: There is no current AL context.

`ALUT_ERROR_AL_ERROR_ON_ENTRY`
: There was already an AL error on entry to `alutCreateBufferFromFile`.

`ALUT_ERROR_ALC_ERROR_ON_ENTRY`
: There was already an ALC error on entry to `alutCreateBufferFromFile`.

`ALUT_ERROR_GEN_BUFFERS`
: There was an error generating an AL buffer.

`ALUT_ERROR_BUFFER_DATA`
: There was an error passing buffer data to AL.

`ALUT_ERROR_IO_ERROR`
: I/O error, consult `errno` for more details.

`ALUT_ERROR_UNSUPPORTED_FILE_TYPE`
: Unsupported file type.

`ALUT_ERROR_UNSUPPORTED_FILE_SUBTYPE`
: Unsupported mode within an otherwise usable file type.

`ALUT_ERROR_CORRUPT_OR_TRUNCATED_DATA`
: The sound data was corrupt or truncated.

alutCreateBufferFromFileImage

Name

`alutCreateBufferFromFileImage` - load in-memory sound data into an OpenAL buffer

Synopsis

```
ALuint alutCreateBufferFromFileImage (const ALvoid *data, ALsizei
length);
```

`alutCreateBufferFromFileImage` tries to guess the sound data format by looking at the contents of the memory region given as parameters and loads the sound data into an OpenAL buffer.

Return Value

On success, `alutCreateBufferFromFileImage` returns a handle to an OpenAL buffer containing the loaded sound. It returns `AL_NONE` on failure.

Errors

`ALUT_ERROR_OUT_OF_MEMORY`
: ALUT ran out of memory.

`ALUT_ERROR_INVALID_OPERATION`
: ALUT has not been initialised.

`ALUT_ERROR_NO_CURRENT_CONTEXT`
: There is no current AL context.

`ALUT_ERROR_AL_ERROR_ON_ENTRY`
: There was already an AL error on entry to `alutCreateBufferFromFileImage`.

`ALUT_ERROR_ALC_ERROR_ON_ENTRY`
: There was already an ALC error on entry to `alutCreateBufferFromFileImage`.

`ALUT_ERROR_GEN_BUFFERS`
: There was an error generating an AL buffer.

`ALUT_ERROR_BUFFER_DATA`
: There was an error passing buffer data to AL.

`ALUT_ERROR_UNSUPPORTED_FILE_TYPE`
: Unsupported file type.

`ALUT_ERROR_UNSUPPORTED_FILE_SUBTYPE`
: Unsupported mode within an otherwise usable file type.

`ALUT_ERROR_CORRUPT_OR_TRUNCATED_DATA`
: The sound data was corrupt or truncated.

alutCreateBufferHelloWorld

Name

`alutCreateBufferHelloWorld` - create a buffer with a 'Hello, world!' sound

Synopsis

```
ALuint alutCreateBufferHelloWorld (void);
```

Description

`alutCreateBufferHelloWorld` returns a handle to an OpenAL buffer containing the sound of someone saying 'Hello, world!'.

Return Value

On success, `alutCreateBufferHelloWorld` returns a handle to an OpenAL buffer containing a 'Hello, world!' sound. It returns `AL_NONE` on failure.

Errors

`ALUT_ERROR_OUT_OF_MEMORY`
: ALUT ran out of memory.

`ALUT_ERROR_INVALID_OPERATION`
: ALUT has not been initialised.

`ALUT_ERROR_NO_CURRENT_CONTEXT`
: There is no current AL context.

`ALUT_ERROR_AL_ERROR_ON_ENTRY`
: There was already an AL error on entry to `alutCreateBufferHelloWorld`.

`ALUT_ERROR_ALC_ERROR_ON_ENTRY`
: There was already an ALC error on entry to `alutCreateBufferHelloWorld`.

`ALUT_ERROR_GEN_BUFFERS`
: There was an error generating an AL buffer.

`ALUT_ERROR_BUFFER_DATA`
: There was an error passing buffer data to AL.

alutCreateBufferWaveform

Name

`alutCreateBufferWaveform` - create a buffer with a synthesized waveform sound

Synopsis

```
ALuint alutCreateBufferWaveform (ALenum waveshape,
                                 ALfloat frequency,
                                 ALfloat phase,
                                 ALfloat duration);
```

Description

`alutCreateBufferWaveform` returns a handle to an OpenAL buffer containing a snippet of audio with the specified waveshape at the specified frequency (in

Hertz), phase (in degrees: -180 to +180) and duration (in seconds). Allowed waveforms are:

- ALUT_WAVEFORM_SINE
- ALUT_WAVEFORM_SQUARE
- ALUT_WAVEFORM_SAWTOOTH
- ALUT_WAVEFORM_WHITENOISE
- ALUT_WAVEFORM_IMPULSE

The duration will always be rounded up to an exact number of cycles of the sound to avoid a click if you loop the sample. The frequency and phase arguments are ignored for ALUT_WHITENOISE.

Return Value

On success, alutCreateBufferWaveform returns a handle to an OpenAL buffer containing the synthesized waveform. It returns AL_NONE on failure.

Errors

ALUT_ERROR_OUT_OF_MEMORY
: ALUT ran out of memory.

ALUT_ERROR_INVALID_ENUM
: An invalid waveform token was given to alutCreateBufferWaveform.

ALUT_ERROR_INVALID_VALUE
: The frequency was not positive or the duration was negative.

ALUT_ERROR_INVALID_OPERATION
: ALUT has not been initialised.

ALUT_ERROR_NO_CURRENT_CONTEXT
: There is no current AL context.

ALUT_ERROR_AL_ERROR_ON_ENTRY
: There was already an AL error on entry to alutCreateBufferWaveform.

ALUT_ERROR_ALC_ERROR_ON_ENTRY
: There was already an ALC error on entry to alutCreateBufferWaveform.

ALUT_ERROR_GEN_BUFFERS
: There was an error generating an AL buffer.

ALUT_ERROR_BUFFER_DATA
: There was an error passing buffer data to AL.

alutLoadMemoryFromFile

Name

alutLoadMemoryFromFile - load a sound file into OpenAL-like data

Synopsis

```
ALvoid *alutLoadMemoryFromFile (const char *filename,
```

```
                                ALenum *format,
                                ALsizei *size,
                                ALfloat *frequency);
```

`alutLoadMemoryFromFile` tries to guess the sound data format by looking at the filename and/or the file contents and loads the sound data into a newly `malloc`ed buffer, possibly converting it in the process. The format is guaranteed to be a standard OpenAL format afterwards.

Return Value

On success, `alutLoadMemoryFromFile` returns a pointer to a newly allocated memory area containing the sound data, which can be `free`d if not needed anymore. It returns `NULL` on failure. If any of the format, size or frequency parameters are non-`NULL`, the respective information about the sound will be passed back.

Errors

`ALUT_ERROR_OUT_OF_MEMORY`
: ALUT ran out of memory.

`ALUT_ERROR_INVALID_OPERATION`
: ALUT has not been initialised.

`ALUT_ERROR_NO_CURRENT_CONTEXT`
: There is no current AL context.

`ALUT_ERROR_AL_ERROR_ON_ENTRY`
: There was already an AL error on entry to `alutLoadMemoryFromFile`.

`ALUT_ERROR_ALC_ERROR_ON_ENTRY`
: There was already an ALC error on entry to `alutLoadMemoryFromFile`.

`ALUT_ERROR_IO_ERROR`
: I/O error, consult `errno` for more details.

`ALUT_ERROR_UNSUPPORTED_FILE_TYPE`
: Unsupported file type.

`ALUT_ERROR_UNSUPPORTED_FILE_SUBTYPE`
: Unsupported mode within an otherwise usable file type.

`ALUT_ERROR_CORRUPT_OR_TRUNCATED_DATA`
: The sound data was corrupt or truncated.

alutLoadMemoryFromFileImage

Name

`alutLoadMemoryFromFileImage` - convert in-memory sound data into OpenAL-like data

Synopsis

```
ALvoid *alutLoadMemoryFromFileImage (const ALvoid *data,
                                     ALsizei length,
                                     ALenum *format,
                                     ALsizei *size,
                                     ALfloat *frequency);
```

`alutLoadMemoryFromFileImage` tries to guess the sound data format by looking at the contents of the memory region given as the first two arguments and loads the sound data into a newly `malloc`ed buffer, possibly converting it in the process. The format is guaranteed to be a standard OpenAL format afterwards.

Return Value

On success, `alutLoadMemoryFromFileImage` returns a pointer to a newly allocated memory area containing the sound data, which can be `free`d if not needed anymore. It returns `NULL` on failure. If any of the format, size or frequency parameters are non-`NULL`, the respective information about the sound will be passed back.

Errors

`ALUT_ERROR_OUT_OF_MEMORY`
: ALUT ran out of memory.

`ALUT_ERROR_INVALID_OPERATION`
: ALUT has not been initialised.

`ALUT_ERROR_NO_CURRENT_CONTEXT`
: There is no current AL context.

`ALUT_ERROR_AL_ERROR_ON_ENTRY`
: There was already an AL error on entry to `alutLoadMemoryFromFileImage`.

`ALUT_ERROR_ALC_ERROR_ON_ENTRY`
: There was already an ALC error on entry to `alutLoadMemoryFromFileImage`.

`ALUT_ERROR_UNSUPPORTED_FILE_TYPE`
: Unsupported file type.

`ALUT_ERROR_UNSUPPORTED_FILE_SUBTYPE`
: Unsupported mode within an otherwise usable file type.

`ALUT_ERROR_CORRUPT_OR_TRUNCATED_DATA`
: The sound data was corrupt or truncated.

alutLoadMemoryHelloWorld

Name

`alutLoadMemoryHelloWorld` - load a 'Hello, world!' sound into OpenAL-like data

Synopsis

```
ALvoid *alutLoadMemoryHelloWorld (ALenum *format,
                                  ALsizei *size,
                                  ALfloat *frequency);
```

Description

`alutLoadMemoryHelloWorld` loads the sound of someone saying 'Hello, world!' into a newly `malloc`ed buffer. The sound data is guaranteed to be in a standard OpenAL format, with a sample frequency chosen by the ALUT implementation.

Return Value

On success, `alutLoadMemoryHelloWorld` returns a pointer to a newly allocated memory area containing the sound data, which can be `free`d if not needed anymore. It returns `NULL` on failure. If any of the format, size or frequency parameters are non-`NULL`, the respective information about the sound will be passed back.

Errors

`ALUT_ERROR_OUT_OF_MEMORY`
: ALUT ran out of memory.

`ALUT_ERROR_INVALID_OPERATION`
: ALUT has not been initialised.

`ALUT_ERROR_NO_CURRENT_CONTEXT`
: There is no current AL context.

`ALUT_ERROR_AL_ERROR_ON_ENTRY`
: There was already an AL error on entry to `alutLoadMemoryHelloWorld`.

`ALUT_ERROR_ALC_ERROR_ON_ENTRY`
: There was already an ALC error on entry to `alutLoadMemoryHelloWorld`.

alutLoadMemoryWaveform

Name

`alutLoadMemoryWaveform` - load a synthesized waveform sound into OpenAL-like data

Synopsis

```
ALvoid *alutLoadMemoryWaveform (ALenum waveshape,
                                ALfloat frequency,
                                ALfloat phase,
                                ALfloat duration,
                                ALenum *format,
                                ALsizei *size,
```

```
                                        ALfloat *sampleFrequency);
```

Description

`alutLoadMemoryWaveform` loads a snippet of audio with the specified waveshape at the specified frequency (in Hertz), phase (in degrees: -180 to +180) and duration (in seconds) into a newly `malloc`ed buffer. The sound data is guaranteed to be in a standard OpenAL format, with a sample frequency chosen by the ALUT implementation. Allowed waveforms are:

- `ALUT_WAVEFORM_SINE`
- `ALUT_WAVEFORM_SQUARE`
- `ALUT_WAVEFORM_SAWTOOTH`
- `ALUT_WAVEFORM_WHITENOISE`
- `ALUT_WAVEFORM_IMPULSE`

The duration will always be rounded up to an exact number of cycles of the sound to avoid a click if you loop the sample. The frequency and phase arguments are ignored for `ALUT_WHITENOISE`.

Return Value

On success, `alutLoadMemoryWaveform` returns a pointer to a newly allocated memory area containing the sound data, which can be `free`d if not needed anymore. It returns `NULL` on failure. If any of the format, size or sample frequency parameters are non-`NULL`, the respective information about the sound will be passed back.

Errors

`ALUT_ERROR_OUT_OF_MEMORY`
: ALUT ran out of memory.

`ALUT_ERROR_INVALID_ENUM`
: An invalid waveform token was given to `alutLoadMemoryWaveform`.

`ALUT_ERROR_INVALID_VALUE`
: The frequency was not positive or the duration was negative.

`ALUT_ERROR_INVALID_OPERATION`
: ALUT has not been initialised.

`ALUT_ERROR_NO_CURRENT_CONTEXT`
: There is no current AL context.

`ALUT_ERROR_AL_ERROR_ON_ENTRY`
: There was already an AL error on entry to `alutLoadMemoryWaveform`.

`ALUT_ERROR_ALC_ERROR_ON_ENTRY`
: There was already an ALC error on entry to `alutLoadMemoryWaveform`.

alutGetMIMETypes

Name

`alutGetMIMETypes` - get list support supported audio MIME types

Synopsis

```
const char *alutGetMIMETypes (ALenum loader);
```

Description

`alutGetMIMETypes` returns a comma-separated list of supported MIME types for the given loader type, e.g. something like "`audio/basic,audio/mpeg,audio/x-wav`". Allowed loader types are:

`ALUT_LOADER_BUFFER`
: For the loaders returning sound data in an OpenAL buffer, e.g. `alutCreateBufferFromFile` and `alutCreateBufferFromFileImage`

`ALUT_LOADER_MEMORY`
: For the loaders returning sound data in a newly allocated memory region, e.g. `alutLoadMemoryFromFile` and `alutLoadMemoryFromFileImage`

It is possible that `ALUT_LOADER_MEMORY` loaders will be unable to support some file types that `ALUT_LOADER_BUFFER` loaders can support (although the reverse is never the case). Furthermore, it is possible that for some file types (notably audio/x-wav) the support may be only for a few sub-formats. For example, an implementation may advertise that audio/x-wav is supported when in fact it only supports uncompressed (i.e. PCM) WAV files and not any of the compressed subformats. In this event, the various ALUT loaders may return an error and set `ALUT_ERROR_UNSUPPORTED_FILE_SUBTYPE` rather than `ALUT_ERROR_UNSUPPORTED_FILE_TYPE` which would indicate that no files of this type are allowed.

Return Value

On success, `alutGetMIMETypes` returns a zero-terminated string which contains a comma-separated list of supported MIME types. It returns `NULL` on failure.

Errors

`ALUT_ERROR_INVALID_ENUM`
: `alutGetMIMETypes` was given an invalid loader token.

`ALUT_ERROR_INVALID_OPERATION`
: ALUT has not been initialised.

`ALUT_ERROR_NO_CURRENT_CONTEXT`
: There is no current AL context.

`ALUT_ERROR_AL_ERROR_ON_ENTRY`

There was already an AL error on entry to `alutGetMIMETypes`.

`ALUT_ERROR_ALC_ERROR_ON_ENTRY`

There was already an ALC error on entry to `alutGetMIMETypes`.

Deprecated WAV loaders

For backwards-compatibility with ALUT 0.x.x, ALUT still offers the three deprecated functions below. Note that on MacOS 0.x.x version, the '`loop`' parameter is omitted from both loader functions.

```
void alutLoadWAVFile (ALbyte *filename,
                      ALenum *format,
                      void **data,
                      ALsizei *size,
                      ALsizei *frequency,
                      ALboolean *loop);

void alutLoadWAVMemory (ALbyte *buffer,
                        ALenum *format,
                        void **data,
                        ALsizei *size,
                        ALsizei *frequency,
                        ALboolean *loop);

void alutUnloadWAV (ALenum format ALvoid *data, ALsizei size, ALsizei
frequency);
```

Version Checking

ALUT version numbers consist of a major version number, a minor version number, and a patchlevel. The former two numbers will match the major/minor version number of the corresponding ALUT specification document and can be accessed at compile time as well as runtime. The patchlevel is not programmatically available and it is incremented only when fixing bugs without any API changes.

alutGetMajorVersion

Name

`alutGetMajorVersion` - return the major ALUT version number

Synopsis

```
ALint alutGetMajorVersion (void);
```

Description

`alutGetMajorVersion` returns the major version number of the ALUT in use, which will match the major version number of the corresponding ALUT specification document.

Return Value

`alutGetMajorVersion` returns the major version number of the ALUT in use.

Errors

`alutGetMajorVersion` can be called in any ALUT state and will never fail.

alutGetMinorVersion

Name

`alutGetMinorVersion` - return the minor ALUT version number

Synopsis

```
ALint alutGetMinorVersion (void);
```

Description

`alutGetMinorVersion` returns the minor version number of the ALUT in use, which will match the minor version number of the corresponding ALUT specification document.

Return Value

`alutGetMinorVersion` returns the minor version number of the ALUT in use.

Errors

`alutGetMinorVersion` can be called in any ALUT state and will never fail.

Compile Time Version Checking

```
#define ALUT_API_MAJOR_VERSION 1

#define ALUT_API_MINOR_VERSION 1
```

Version 1.0.0 introduced the above preprocessor symbols, whose values will be incremented appropriately in future revisions of ALUT. In version 1.1.0,

`alutLoadMemoryHelloWorld` and `alutLoadMemoryWaveform` have been added to the ALUT API.

Example: Version consistency check

Applications can verify at runtime that they have been compiled and linked with the matching header file and library file as follows:

```
#ifdef ALUT_API_MAJOR_VERSION
if (alutGetMajorVersion () != ALUT_API_MAJOR_VERSION ||
    alutGetMinorVersion () != ALUT_API_MINOR_VERSION)
  /* Oh-oh!  The ALUT header and the ALUT library are different
revisions... */
#else
  /* Oh-oh!  We're linking against an ALUT 0.x.x header file... */
#endif
```

Sleeping

Having a general utility function like the following in an audio-related toolkit might seem strange at first, but sleeping is a common task in a lot of audio demos and it can't be done portably without cluttering the source code with `#ifdefs`.

alutSleep

Name

`alutSleep` - sleep for a given number of seconds

Synopsis

```
ALboolean alutSleep (ALfloat duration);
```

Description

`alutSleep` will delay the execution of the current thread for at least the given amount of seconds. It will only return earlier if a signal has been delivered to the thread, but this does not count as an error. Note that sleeping for zero seconds will give other runnable threads a chance to run.

Return Value

`alutSleep` returns `AL_TRUE` on success or `AL_FALSE` on failure. Note that current implementations will always succeed if the duration is non-negative, but this might change in the future.

Errors

`ALUT_ERROR_INVALID_VALUE`
: `alutSleep` was given a negative duration.