ObjectAL for iPhone

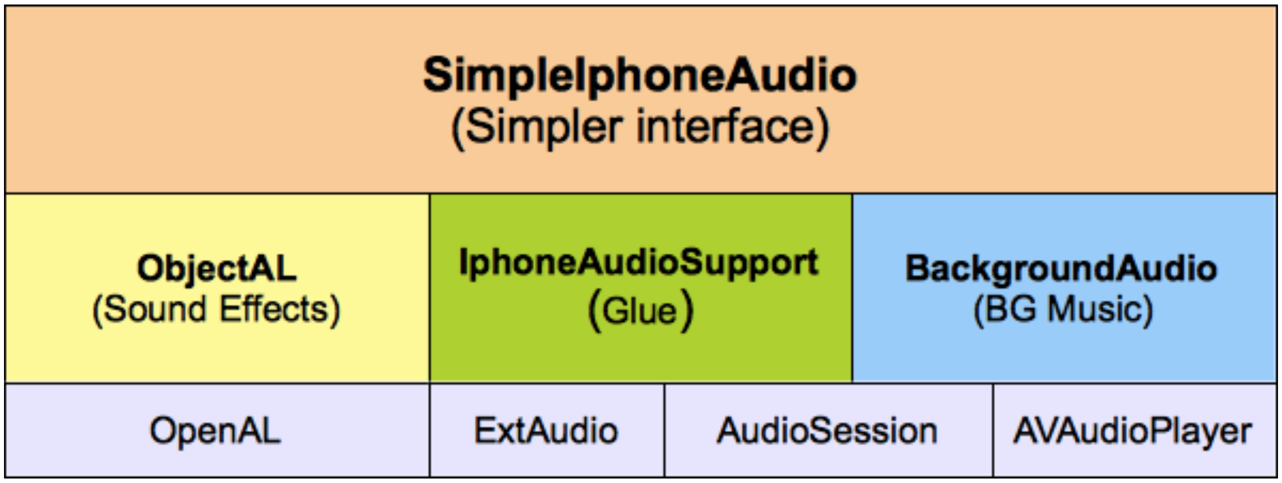
Version 1.0

Contents

- **Introduction**
- **ObjectAL and OpenAL**
- **Audio Formats**
- **Adding ObjectAL to your project** (also, installing the documentation into XCode)
- **Using SimpleIphoneAudio**
- **Using ObjectAL and BackgroundAudio**
- **Other Examples**
- **Simulator Issues**

Introduction

ObjectAL for iPhone is designed to be a simpler, more intuitive interface to OpenAL and AVAudioPlayer. There are four main parts to **ObjectAL for iPhone**:



- **ObjectAL** gives you full access to the OpenAL system without the hassle of the C API. All OpenAL operations can be performed using first class objects and properties, without needing to muddle around with arrays of data, maintain IDs, or pass around pointers to basic types.

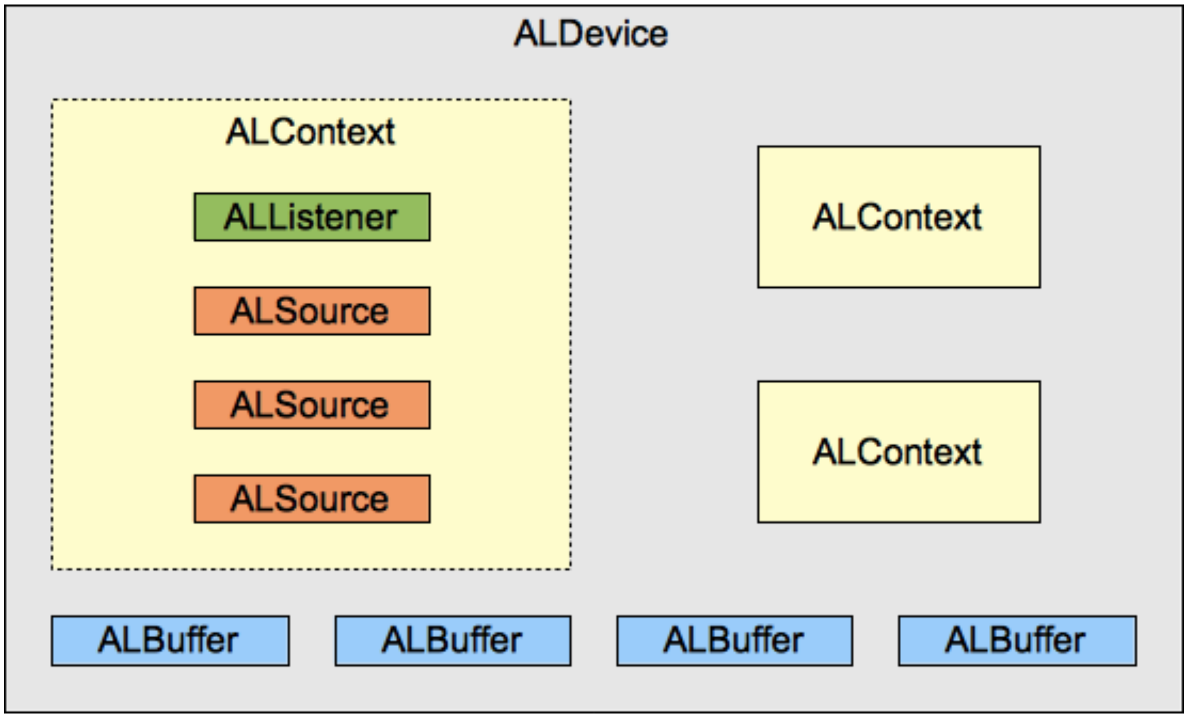
- **BackgroundAudio** provides a simpler interface to AVAudioPlayer, allowing you to play, stop, pause, and mute background music tracks. As well, it provides an easy way to configure how AVAudioPlayer will handle iPod-style music playing and the silent switch.

- **IphoneAudioSupport** provides support functionality for audio in iPhone, including automatic interrupt handling and audio data loading routines.

- **SimpleIphoneAudio** layers on top of the other three, providing an even simpler interface for playing background music and sound effects.

ObjectAL and OpenAL

ObjectAL follows the same basic principles as the OpenAL API (http://connect.creativelabs.com/openal) .



- **ObjectAL** provides some overall controls that affect everything, and manages the current context.
- **ALDevice** represents a physical audio device.
  Each device can have one or more contexts (**ALContext**) created on it, and can have multiple buffers (**ALBuffer**) associated with it.
- **ALContext** controls the overall sound environment, such as distance model, doppler effect, and speed of sound.
  Each context has one listener (**ALListener**), and can have multiple sources (**ALSource**) opened on it (up to a maximum of 32 overall on iPhone).
- **ALListener** represents the listener of sounds originating on its context (one listener per context). It has position, orientation, and velocity.
- **ALSource** is a sound emitting source that plays sound data from an **ALBuffer**. It has position, direction, velocity, as well as other properties which determine how the sound is emitted.
- **ChannelSource** allows you to reserve a certain number of sources for special purposes.
- **ALBuffer** is simply a container for sound data. Only linear PCM is supported directly, but **IphoneAudioSupport** load methods, and **SimpleIphoneAudio** effect preload and play methods, will automatically convert any formats that don't require hardware decoding (though conversion results in a longer loading time).

Further information regarding the more advanced features of OpenAL (such as distance models) are available via the OpenAL Documentation at Creative Labs.
In particular, read up on the various property values for sources and listeners (such as Doppler Shift) in the **OpenAL Programmer's Guide**, and distance models in section 3 of the **OpenAL Specification**.
Also be sure to read the OpenAL FAQ from Apple.

Audio Formats

According to the OpenAL FAQ from Apple:

- To use OpenAL for playback, your application typically reads audio data from disk using Extended Audio File Services. In this process you convert the on-disk format, as needed, into one of the OpenAL playback formats (**IphoneAudioSupport** and **SimpleIphoneAudio** do this for you).
- The on-disk audio format that your application reads must be PCM (uncompressed) or a compressed format that does not use hardware decompression, such as IMA-4.
- The supported playback formats for OpenAL in iPhone OS are identical to those for OpenAL in Mac OS X. You can play the following linear PCM variants: mono 8-bit, mono 16-bit, stereo 8-bit, and stereo 16-bit.

BackgroundAudio supports all hardware and software decoded formats as specified by Apple here.

Adding ObjectAL to your project

To add ObjectAL to your project, do the following:

1. Copy libs/ObjectAL from this project into your project. You can simply drag it into the "Groups & Files" section in xcode if you like (be sure to select "Copy items into destination group's folder"). Alternatively, you can build ObjectAL as a static library (as it's configured to do in this project).
2. Add the following frameworks to your project:
   - OpenAL.framework
   - AudioToolbox.framework
   - AVFoundation.framework
3. Start using ObjectAL!

Note: The demos in this project use Cocos2d, a very nice 2d game engine. However, ObjectAL doesn't require it. You can just as easily use ObjectAL in your Cocoa app or anything you wish.

Note #2: You do NOT have to provide a link to the Apache license from within your application. Simply including a copy of the license in your project is sufficient.

Installing the ObjectAL Documentation into XCode

You can install the ObjectAL documentation into XCode's Developer Documentation system by doing the following:

1. Install Doxygen
2. Build the "ObjectAL Doc" target in this project.
3. Open the developer documentation and type "ObjectAL" into the search box.

Using SimpleIphoneAudio

By far, the easiest component to use is **SimpleIphoneAudio**. You sacrifice some power for ease-of-use, but for many projects it is more than sufficient.

Here is a code example:

```objc
// SomeClass.h
@interface SomeClass : NSObject
{
    // No objects to keep track of...
}

@end


// SomeClass.m
#import "SomeClass.h"
#import "SimpleIphoneAudio.h"

#define SHOOT_SOUND @"shoot.caf"
#define EXPLODE_SOUND @"explode.caf"

#define INGAME_MUSIC_FILE @"bg_music.mp3"
#define GAMEOVER_MUSIC_FILE @"gameover_music.mp3"

@implementation SomeClass

- (id) init
{
    if(nil != (self = [super init]))
    {
        // We don't want ipod music to keep playing since
        // we have our own bg music.
        [SimpleIphoneAudio sharedInstance].allowIpod = NO;

        // Mute all audio if the silent switch is turned on.
        [SimpleIphoneAudio sharedInstance].honorSilentSwitch = YES;

        // This loads the sound effects into memory so that
        // there's no delay when we tell it to play them.
        [[SimpleIphoneAudio sharedInstance] preloadEffect:SHOOT_SOUND];
        [[SimpleIphoneAudio sharedInstance] preloadEffect:EXPLODE_SOUND];
    }
    return self;
}

- (void) dealloc
{
    // You might call stopEverything here depending on your needs
    //[[SimpleIphoneAudio sharedInstance] stopEverything];

    [super dealloc];
}

- (void) onGameStart
{
    // Play the BG music and loop it.
    [[SimpleIphoneAudio sharedInstance] playBg:INGAME_MUSIC_FILE loop:YES];
}

- (void) onGamePause
{
    [SimpleIphoneAudio sharedInstance].bgPaused = YES;
    [SimpleIphoneAudio sharedInstance].muted = YES;
}

- (void) onGameResume
{
    [SimpleIphoneAudio sharedInstance].muted = NO;
    [SimpleIphoneAudio sharedInstance].bgPaused = NO;
}

- (void) onGameOver
{
    // Could use stopEverything here if you want
    [[SimpleIphoneAudio sharedInstance] stopAllEffects];

    // We only play the game over music through once.
    [[SimpleIphoneAudio sharedInstance] playBg:GAMEOVER_MUSIC_FILE];
}

- (void) onShipShotABullet
{
    [[SimpleIphoneAudio sharedInstance] playEffect:SHOOT_SOUND];
}

- (void) onShipGotHit
{
    [[SimpleIphoneAudio sharedInstance] playEffect:EXPLODE_SOUND];
}

- (void) onQuitToMainMenu
{
    // Stop all music and sound effects.
    [[SimpleIphoneAudio sharedInstance] stopEverything];

    // Unload all sound effects and bg music so that it doesn't fill
    // memory unnecessarily.
    [[SimpleIphoneAudio sharedInstance] unloadAll];
}

@end
```

Using ObjectAL and BackgroundAudio

ObjectAL and **BackgroundAudio** offer you much more power, but at the cost of complexity. Here's the same thing as above, done using **ObjectAL** and **BackgroundAudio** directly:

```
// SomeClass.h
#import "ObjectAL.h"

@interface SomeClass : NSObject
{
    ALDevice* device;
    ALContext* context;
    ChannelSource* channel;
    ALBuffer* shootBuffer;
    ALBuffer* explosionBuffer;
}

@end


// SomeClass.m
#import "SomeClass.h"
#import "BackgroundAudio.h"
#import "IphoneAudioSupport.h"

#define SHOOT_SOUND @"shoot.caf"
#define EXPLODE_SOUND @"explode.caf"

#define INGAME_MUSIC_FILE @"bg_music.mp3"
#define GAMEOVER_MUSIC_FILE @"gameover_music.mp3"

@implementation SomeClass

- (id) init
{
    if(nil != (self = [super init]))
    {
        // Create the device and context.
        device = [[ALDevice deviceWithDeviceSpecifier:nil] retain];
        context = [[ALContext contextOnDevice:device attributes:nil] retain];
        [ObjectAL sharedInstance].currentContext = context;

        // Deal with interruptions for me!
        [IphoneAudioSupport sharedInstance].handleInterruptions = YES;

        // We don't want ipod music to keep playing since
        // we have our own bg music.
        [BackgroundAudio sharedInstance].allowIpod = NO;

        // Mute all audio if the silent switch is turned on.
        [BackgroundAudio sharedInstance].honorSilentSwitch = YES;

        // Take all 32 sources for this channel.
        // (we probably won't use that many but what the heck!)
        channel = [[ChannelSource channelWithSources:32] retain];

        // Preload the buffers so we don't have to load and play them later.
        shootBuffer = [[[IphoneAudioSupport sharedInstance]
                        bufferFromFile:SHOOT_SOUND] retain];
        explosionBuffer = [[[IphoneAudioSupport sharedInstance]
                            bufferFromFile:EXPLODE_SOUND] retain];
    }
    return self;
}

- (void) dealloc
{
    [channel release];
    [shootBuffer release];
    [explosionBuffer release];

    // Note: You'll likely only have one device and context open throughout
    // your program, so in a real program you'd be better off making a
    // singleton object that manages the device and context, rather than
    // allocating/deallocating it here.
    // Another important thing to note: The simulator is a bit weird,
    // and will freeze for 60+ seconds if you close the current context
    // after BackgroundAudio WAS playing but is now NOT playing!
    // This doesn't happen on a real device, though.
    [context release];
    [device release];

    [[BackgroundAudio sharedInstance] stop];

    [super dealloc];
}

- (void) onGameStart
{
    // Play the BG music and loop it forever.
    [BackgroundAudio sharedInstance].numberOfLoops = -1;
    [[BackgroundAudio sharedInstance] playFile:INGAME_MUSIC_FILE];
}

- (void) onGamePause
{
    [BackgroundAudio sharedInstance].paused = YES;
    channel.muted = YES;
}

- (void) onGameResume
{
    channel.muted = NO;
    [BackgroundAudio sharedInstance].paused = NO;
}

- (void) onGameOver
{
    [channel stop];
    [[BackgroundAudio sharedInstance] stop];

    // We only play the game over music through once.
    [BackgroundAudio sharedInstance].numberOfLoops = 0;
    [[BackgroundAudio sharedInstance] playFile:GAMEOVER_MUSIC_FILE];
}

- (void) onShipShotABullet
{
    [channel play:shootBuffer];
}
```

```
- (void) onShipGotHit
{
    [channel play:explosionBuffer];
}

- (void) onQuitToMainMenu
{
    // Stop all music and sound effects.
    [channel stop];
    [[BackgroundAudio sharedInstance] stop];
}

@end
```

Other Examples

The demo scenes in this distribution have been crafted to demonstrate common uses of this library. Try them out and go through the code to see how it's done. I've done my best to keep the code readable. Really!

The current demos are:

- **ChannelsDemo**: Demonstrates using audio channels.
- **CrossFadeDemo**: Demonstrates crossfading between two sources.
- **PlanetKillerDemo**: Demonstrates using **SimpleIphoneAudio** in a game setting.
- **SingleSourceDemo**: Demonstrates using a location based source and a listener.
- **TwoSourceDemo**: Demonstrates using two location based sources and a listener.
- **VolumePitchPanDemo**: Demonstrates using gain, pitch, and pan controls.

Simulator Issues

As you've likely heard time and time again, the simulator is no substitute for the real thing. The simulator is buggy. It can run faster or slower than a real device. It fails system calls that a real device doesn't. It shows graphics glitches that a real device doesn't. Sounds stop working. Dogs and cats living together, etc, etc. When things look wrong, try it on a real device before bugging people.

Error Codes on the Simulator

When using **BackgroundAudio** (or **SimpleIphoneAudio**) to play BG music, you will see the following error in the debug output, which can safely be ignored:

- MyProject[39621:207] Error: **BackgroundAudio**: Unknown session error (error code ffffffce)

Other weird errors can crop up from time to time as well. If you're unsure, run it on a real device to see if it still occurs.

Playback Issues

The simulator is notoriously finicky when it comes to audio playback. Any number of programs you've installed on your mac can cause the simulator to stop playing bg music, or effects, or both!

Some things to check when sound stops working:

- Try resetting and restarting the simulator.
- Try restarting XCode, cleaning, and recompiling your project.
- Try rebooting your computer.
- Open "Audio MIDI Setup" (type "midi" into spotlight to find it) and make sure "Built-in Output" is set to 44100.0 Hz.
- Go to System Preferences -> Sound -> Output, and ensure that "Play sound effects through" is set to "Internal Speakers"
- Go to System Preferences -> Sound -> Input, and ensure that it is using internal sound devices.
- Go to System Preferences -> Sound -> Sound Effects, and ensure "Play user interface sound effects" is checked.
- Some codecs may cause problems with sound playback. Try removing them.
- Programs that redirect audio can wreak havoc on the simulator. Try removing them.

Simulator Freezups

There's a particularly nasty bug in the simulator's OpenAL and AVAudioPlayer implementation that causes the simulator to freeze for 60+ seconds in a very specific case:

If you use **BackgroundAudio** to play background music, then stop the music, then close the current context, the simulator will freeze (a real device won't).

This is not really a huge problem, however, since you really should be making a sound manager singleton object (what **SimpleIphoneAudio** is, basically) to handle the **ALDevice** and **ALContext** (which will in 99.9% of cases last for the entire duration of your program). You can have a look inside **SimpleIphoneAudio** "dealloc" method to see how to avoid this problem in case you really do need to deallocate a device.
