USER GUIDE FOR cJMAP SPEECH ENHANCEMENT IMPLEMENTATION ON *i*OS PLATFORM FOR HEARING AID APPLICATIONS

Objective C shell for iOS app development using either superpowered or core audio

Chandan K A Reddy, Nikhil Shankar, Issa Panahi
STATISTICAL SIGNAL PROCESSING RESEARCH LABORATORY (SSPRL)
UNIVERSITY OF TEXAS AT DALLAS

MARCH 2017

This work was supported by the National Institute of the Deafness and Other Communication Disorders (NIDCD) of the National Institutes of Health (NIH) under the award number 1R01DC015430-01. The content is solely the responsibility of the authors and does not necessarily represent the official views of the NIH.

Table of Contents

INTRODUCTION

The cJMAP app is designed for noise reduction and speech enhancement in real time. The contents of this user guide gives you the steps to implement the cJMAP algorithm on iOS devices (iPhone and iPad) and the steps to be followed after installing the app on the smartphone. This app will be an open source and portable research platform for hearing improvement studies.

This user guide covers the software tools required for implementing the algorithm, how to run C codes on iOS devices and usage of other tools that are quite helpful in creating audio apps for audio playback in real time.

The C codes used for the cJMAP algorithm are made available publicly on the following website: http://www.utdallas.edu/ssprl/hearing-aid-project/
The codes can be accessed and used with proper consent of the author for further improvements in research activities related to hearing aids.

The screenshot of the first look of our app is as shown in Figure 1 below,

Figure 1

1. SOFTWARE TOOLS

iOS is a mobile operating system created and developed by Apple Inc. for devices like iPhones, iPads and iPods. The languages used in this system are C, C++, Objective-C and Swift. All third-party apps are authorized and can be made available through the Apple's app store for devices with iPhone OS 2.0 and higher. The apps must be written in either Swift or Objective-C. We use Objective-C as it has an option of running C or C++ codes within the iOS environment. The codes must be compiled specifically for iOS and the 64-bit ARM architecture (typically using Xcode).

To design apps for iOS devices, a Mac OS is needed. The entire process can be carried out using Xcode IDE (Integrated Development Environment). It is a software package that can be installed for free from the app store. It is not possible to develop iOS apps using a Windows or Linux based computers.

From iOS version 9 and Xcode version 7 onwards, it has become possible to perform app development without the need to enroll or register in the Apple Developer Program. It is worth mentioning that although apps developed can be run on iPhones/iPads, they cannot be published on the App Store without registering as an Apple Developer.

To download the latest version of Xcode

1. Open the App Store app on your Mac (by default it's in the Dock).
2. In the search field in the top-right corner, type Xcode and press the Return key.
   The Xcode app shows up as the first search result.
3. Click Get and then click Install App.
4. Enter your Apple ID and password when prompted.
   Xcode is downloaded into your /Applications directory.

2. BUILD AN IOS APP

2.1 Programming Language

For creating iOS apps, Objective-C is used to create the required shell. Objective-C constitutes a superset of C, allowing one to seamlessly call C functions by just importing a header file. The execution of C codes occurs efficiently within the Objective-C environment due to the absence of any overhead translation or matching.

2.2 Creating Objective-C Shell

The creation of an Objective-C shell starts by creating a GUI to link data to a C code. The steps needed for creating a basic shell are listed below:

- Open Xcode and select "Create a new Xcode project" on the startup splash screen, see Figure 2. If in case, no splash screen comes up, select File->New->Project or press Command + Shift + n.

Figure 2

- On the page that comes up, shown in Figure 3, choose the template of the project as "Single View Application" and click Next.

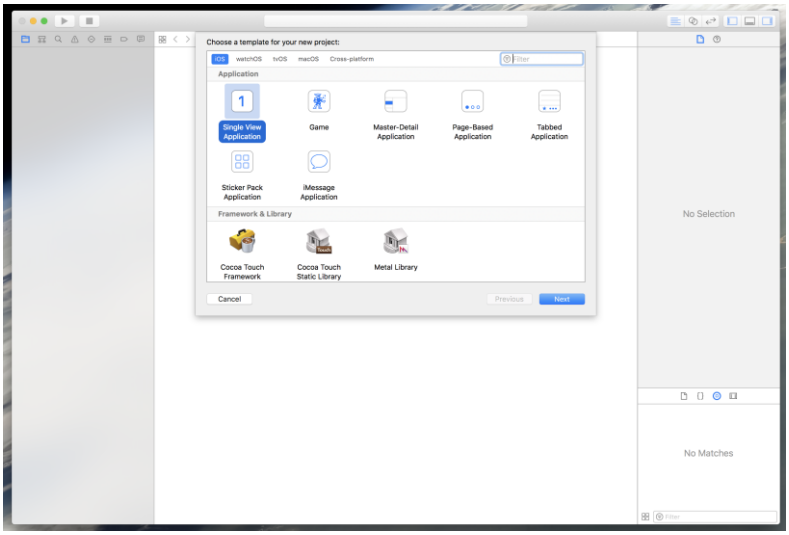


Figure 3

- For options that come up, set the name of the project, select Team as "**None**", set your organization name and the organization identifier. Remember to set the Language to "**Objective-C**" as this option **cannot be reversed**. Select devices as "**Universal**" and deselect all the options.
- Store the project in the Directory of your choice and click **Create**. If desired, deselect the option "**Create Git Repository**".
- To be able to build the app for iPhones, you would need to sign the application and select a Team. In Xcode, select "User Name (Personal Team)" for Team and Xcode will automatically sign the application.

2.3 Adding C File

- To add a C file to the project, navigate to the "JMAP SE" folder in the project navigator and select "New File…", shown in Figure 4.
- In the "iOS" tab, shown in Figure 5, under "Source", select "C File" and click Next.

- On the page that comes up, write the file name and also select the option that states “Also create a header file” and click Next.

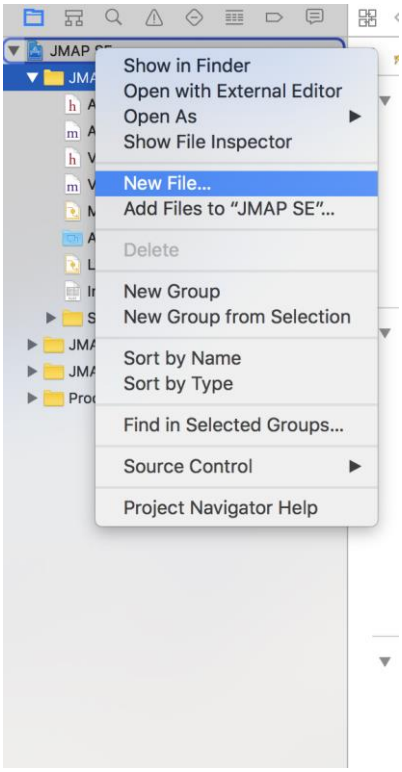


Figure 4

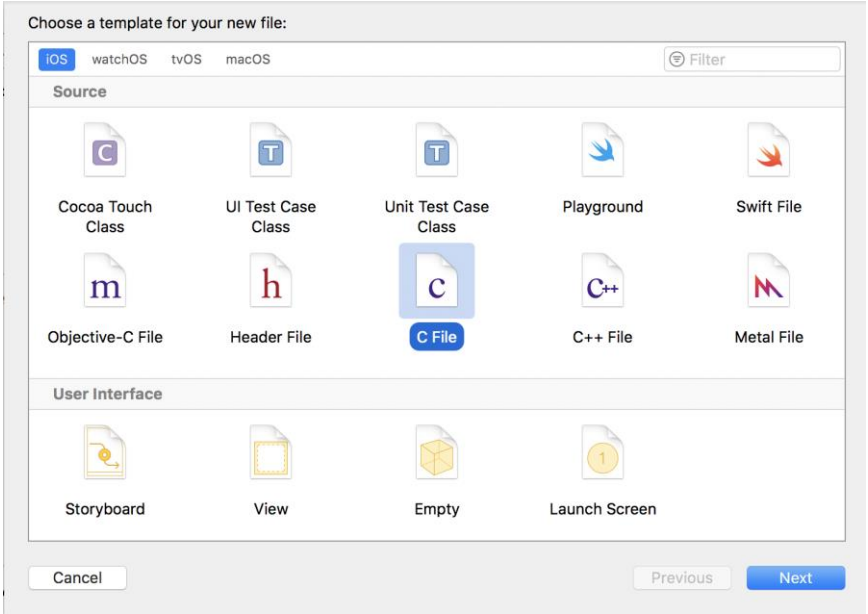


Figure 5

- An existing C code can be directly attached to the app, instead of a writing a new C code. It will be explained in the further topics.
- Once the C code is attached, add the header file in the “View Controller.m” file to use the contents in C code.

3. SUPERPOWERED SDK

- This is a low latency audio SDK for iOS and Android.
- Superpowered accomplishes this using patent-pending DSP optimization technology to achieve desktop grade performance on mobile devices.
- Superpowered technology uses less than half the power of Apple's Core Audio and is more than twice as fast as Apple's vDSP.
- The Superpowered Audio SDK empowers developers to remove CPU resource limitations, and develop cross-platform audio for iOS, Android and wearable devices. It includes:

1. Example apps/projects for iOS, OSX and Android
2. Static library files
3. Decoder for MP3, AAC, WAV, AIFF and STEMS
4. Advanced Audio Player (including time stretching, pitch shifting, resampling, looping, scratching, etc.)
5. HTTP Live Streaming
6. Effects: echo, flanger, gate, reverb, rool, whoosh, 3 band equalizers, biquad IIR filters (low-pass, high-pass, bandpass, high-shelf, low-shelf, parametric, notch)
7. Dynamics: compressor, limiter, clipper.
8. Time Stretching and Pitch Shifting, resampler
9. Polar and Complex FFT
10. Recorder
11. Open-source audio system input/output classes
12. Time-domain to frequency domain class (including inverse)
13. Bandpass filterbank for time-domain frequency analysis
14. Audio analyzer: key detection, bpm detection, beatgrid detection, waveform generation, loudness/peak analysis

15. Stereo and mono mixers
16. Simple audio functions (volume, volume ramp, peak, float-short conversion, interleaving and de-interleaving)

- The code is made as an open source and can be downloaded by going to the following link below,
  http://superpowered.com/
- Once it is downloaded you can go to examples_ios folder as shown in Figure 6, to find various examples provided by superpowered that can be used and modified based on the requirements for our applications.
- When you click on the SuperpoweredFrequencyDomain as shown in Figure 6, the code can be opened on Xcode directly.
- All modifications can be done in "View Controller.mm" file present inside as shown in Figure 7, the data in frequency domain can be modified.

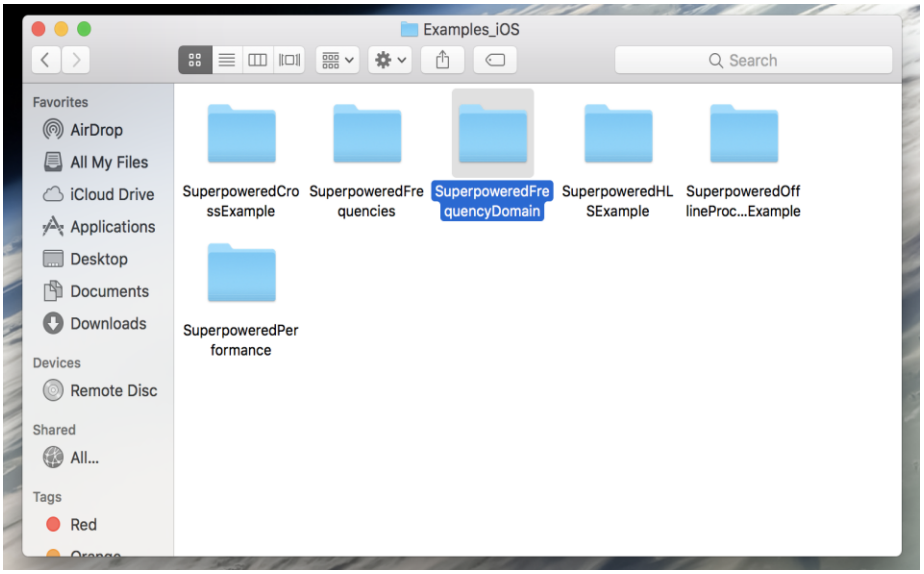


Figure 6

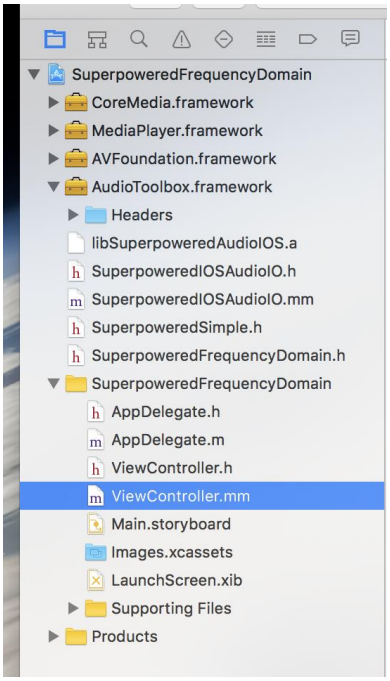


Figure 7

- The input will be considered to be in time domain, then converted to frequency domain and once you are done with the modifications the data will be again converted back to time domain to obtain the output.
- Refer the block diagram shown in Figure 8 for better understanding,

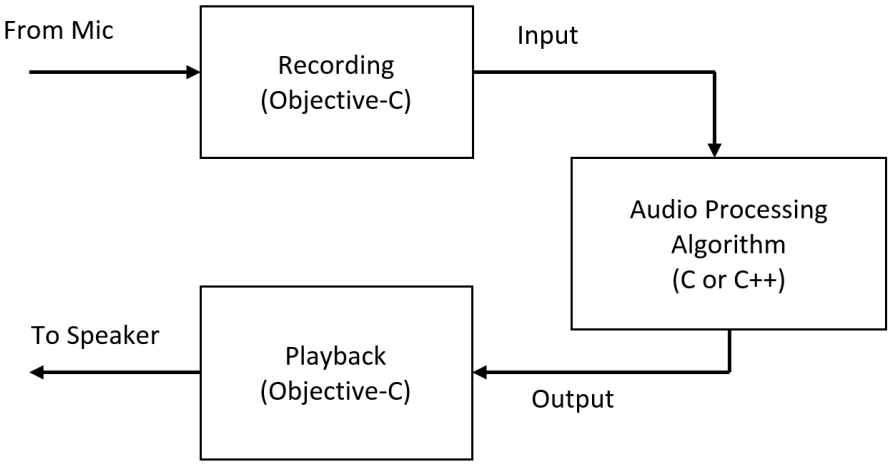


Figure 8

4. CORE AUDIO BY APPLE

- Core Audio is the digital audio infrastructure of iOS and OS X. It consists of numerous software frameworks designed to handle the audio needs in our application.
- Core Audio in iOS is optimized for the computing resources available in a battery-powered mobile platform.
- Most Core Audio services use and manipulate audio in linear pulse-code-modulated (*linear PCM*) format, the most common uncompressed digital audio data format.
- Core audio provides us with many services like,
- Audio Queue Services lets you record, play, pause, loop, and synchronize audio. It employs codecs as necessary to deal with compressed audio formats.
- The AVAudioPlayer class provides a simple Objective-C interface for playing and looping audio as well as implementing rewind and fast-forward.
- For more information related to core audio please refer this link given below.,
  https://developer.apple.com/library/content/documentation/MusicAudio/Conceptual/CoreAudioOverview/WhatisCoreAudio/WhatisCoreAudio.html-//apple_ref/doc/uid/TP40003577-CH3-SW1.
- Steps to be followed before attaching the SE algorithm to core audio are;
  1. Identify the audio component **(kAudioUnitType_Output/ kAudioUnitSubType_RemoteIO/ kAudioUnitManufacturerApple)**
  2. Use **AudioComponentFindNext**(NULL, &descriptionOfAudioComponent) to obtain the AudioComponent, which is like the factory with which you obtain the audio unit
  3. Use **AudioComponentInstanceNew**(ourComponent, &audioUnit) to make an instance of the audio unit
  4. Enable IO for recording and possibly playback with **AudioUnitSetProperty**
  5. Describe the audio format in an **AudioStreamBasicDescription** structure, and apply the format using **AudioUnitSetProperty**
  6. Provide a callback for recording, and possibly playback, again using **AudioUnitSetProperty**

- The code for both recording and playback is as shown in figure 9 and figure 10 respectively. Modifications can be made to them based on the application.

```
//RECORDING

static OSStatus recordingCallback(void *inRefCon,
                                  AudioUnitRenderActionFlags *ioActionFlags,
                                  const AudioTimeStamp *inTimeStamp,
                                  UInt32 inBusNumber,
                                  UInt32 inNumberFrames,
                                  AudioBufferList *ioData) {

    // TODO: Use inRefCon to access our interface object to do stuff
    // Then, use inNumberFrames to figure out how much data is available, and make
    // that much space available in buffers in an AudioBufferList.

    AudioBufferList *bufferList; // <- Fill this up with buffers (you will want to malloc it, as it's a dynamic-length list)

    // Then:
    // Obtain recorded samples

    OSStatus status;

    status = AudioUnitRender([audioInterface audioUnit],
                             ioActionFlags,
                             inTimeStamp,
                             inBusNumber,
                             inNumberFrames,
                             bufferList);
    checkStatus(status);

    // Now, we have the samples we just read sitting in buffers in bufferList
    DoStuffWithTheRecordedAudio(bufferList);
    return noErr;
}
```

Figure 9

```
static OSStatus playbackCallback(void *inRefCon,
                                 AudioUnitRenderActionFlags *ioActionFlags,
                                 const AudioTimeStamp *inTimeStamp,
                                 UInt32 inBusNumber,
                                 UInt32 inNumberFrames,
                                 AudioBufferList *ioData) {
    // Notes: ioData contains buffers (may be more than one!)
    // Fill them up as much as you can. Remember to set the size value in each buffer to match how
    // much data is in the buffer.
    return noErr;
}
```

Figure 10

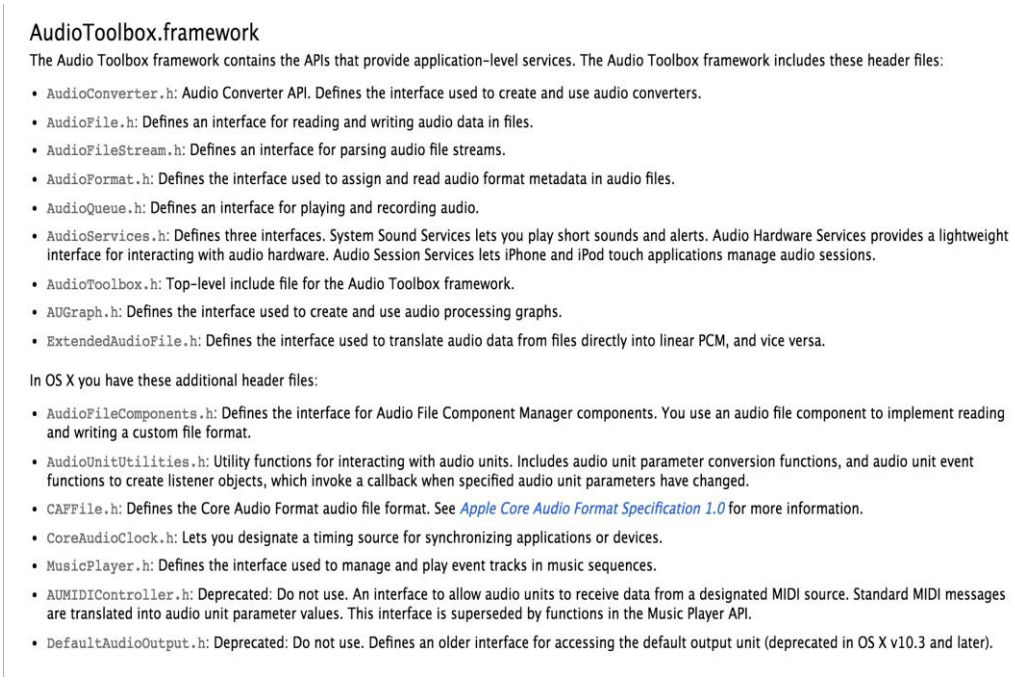
- For initializing, we consider a member variable of type AudioComponentInstance which will contain our audio unit.

- The audio format described below uses SInt16 for samples (i.e. signed, 16 bits per sample) as shown in figure 11.

```
OSStatus status;
AudioComponentInstance audioUnit;

// Describe audio component
AudioComponentDescription desc;
desc.componentType = kAudioUnitType_Output;
desc.componentSubType = kAudioUnitSubType_RemoteIO;
desc.componentFlags = 0;
desc.componentFlagsMask = 0;
desc.componentManufacturer = kAudioUnitManufacturer_Apple;

// Get component
AudioComponent inputComponent = AudioComponentFindNext(NULL, &desc);

// Get audio units
status = AudioComponentInstanceNew(inputComponent, &audioUnit);
checkStatus(status);

// Enable IO for recording
UInt32 flag = 1;
status = AudioUnitSetProperty(audioUnit,
                              kAudioOutputUnitProperty_EnableIO,
                              kAudioUnitScope_Input,
                              kInputBus,
                              &flag,
                              sizeof(flag));
checkStatus(status);

// Enable IO for playback
status = AudioUnitSetProperty(audioUnit,
                              kAudioOutputUnitProperty_EnableIO,
                              kAudioUnitScope_Output,
                              kOutputBus,
                              &flag,
                              sizeof(flag));
checkStatus(status);

// Describe format
audioFormat.mSampleRate         = 44100.00;
audioFormat.mFormatID           = kAudioFormatLinearPCM;
audioFormat.mFormatFlags        = kAudioFormatFlagIsSignedInteger | kAudioFormatFlagIsPacked;
audioFormat.mFramesPerPacket    = 1;
audioFormat.mChannelsPerFrame   = 1;
audioFormat.mBitsPerChannel     = 16;
audioFormat.mBytesPerPacket     = 2;
audioFormat.mBytesPerFrame      = 2;

// Apply format
status = AudioUnitSetProperty(audioUnit,
                              kAudioUnitProperty_StreamFormat,
                              kAudioUnitScope_Output,
                              kInputBus,
                              &audioFormat,
                              sizeof(audioFormat));
checkStatus(status);
status = AudioUnitSetProperty(audioUnit,
                              kAudioUnitProperty_StreamFormat,
                              kAudioUnitScope_Input,
                              kOutputBus,
                              &audioFormat,
                              sizeof(audioFormat));
checkStatus(status);


// Set input callback
AURenderCallbackStruct callbackStruct;
callbackStruct.inputProc = recordingCallback;
callbackStruct.inputProcRefCon = self;
status = AudioUnitSetProperty(audioUnit,
                              kAudioOutputUnitProperty_SetInputCallback,
                              kAudioUnitScope_Global,
                              kInputBus,
                              &callbackStruct,
                              sizeof(callbackStruct));
checkStatus(status);

// Set output callback
callbackStruct.inputProc = playbackCallback;
callbackStruct.inputProcRefCon = self;
status = AudioUnitSetProperty(audioUnit,
                              kAudioUnitProperty_SetRenderCallback,
                              kAudioUnitScope_Global,
                              kOutputBus,
                              &callbackStruct,
                              sizeof(callbackStruct));
checkStatus(status);

// Disable buffer allocation for the recorder (optional - do this if we want to pass in our own)
flag = 0;
status = AudioUnitSetProperty(audioUnit,
                              kAudioUnitProperty_ShouldAllocateBuffer,
                              kAudioUnitScope_Output,
                              kInputBus,
                              &flag,
                              sizeof(flag));

// TODO: Allocate our own buffers if we want
// Initialise
status = AudioUnitInitialize(audioUnit);
checkStatus(status);
```

Figure 11

- Write the SE algorithm as a different function and call it in the main code.
- The above used code is made publicly available and can be accessed from the website, http://atastypixel.com/blog/using-remoteio-audio-unit/ .
- A sample code which is made available by apple was also referred for developing the application. https://developer.apple.com/library/content/samplecode/aurioTouch/Introduction/Intro.html#//apple_ref/doc/uid/DTS40007770 .
- Different frameworks were used for the application mentioned in this user guide.
    1. Audio Toolbox Framework - Record or play audio, convert formats, parse audio streams, and configure your audio session. The Audio Toolbox framework provides interfaces for recording, playback, and stream parsing. In iOS, the framework provides additional interfaces for managing audio sessions. Different header files used in the framework is as shown in figure 12.



AudioToolbox.framework

The Audio Toolbox framework contains the APIs that provide application-level services. The Audio Toolbox framework includes these header files:

- `AudioConverter.h`: Audio Converter API. Defines the interface used to create and use audio converters.
- `AudioFile.h`: Defines an interface for reading and writing audio data in files.
- `AudioFileStream.h`: Defines an interface for parsing audio file streams.
- `AudioFormat.h`: Defines the interface used to assign and read audio format metadata in audio files.
- `AudioQueue.h`: Defines an interface for playing and recording audio.
- `AudioServices.h`: Defines three interfaces. System Sound Services lets you play short sounds and alerts. Audio Hardware Services provides a lightweight interface for interacting with audio hardware. Audio Session Services lets iPhone and iPod touch applications manage audio sessions.
- `AudioToolbox.h`: Top-level include file for the Audio Toolbox framework.
- `AUGraph.h`: Defines the interface used to create and use audio processing graphs.
- `ExtendedAudioFile.h`: Defines the interface used to translate audio data from files directly into linear PCM, and vice versa.

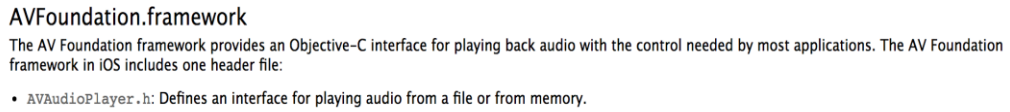
In OS X you have these additional header files:

- `AudioFileComponents.h`: Defines the interface for Audio File Component Manager components. You use an audio file component to implement reading and writing a custom file format.
- `AudioUnitUtilities.h`: Utility functions for interacting with audio units. Includes audio unit parameter conversion functions, and audio unit event functions to create listener objects, which invoke a callback when specified audio unit parameters have changed.
- `CAFFile.h`: Defines the Core Audio Format audio file format. See *Apple Core Audio Format Specification 1.0* for more information.
- `CoreAudioClock.h`: Lets you designate a timing source for synchronizing applications or devices.
- `MusicPlayer.h`: Defines the interface used to manage and play event tracks in music sequences.
- `AUMIDIController.h`: Deprecated: Do not use. An interface to allow audio units to receive data from a designated MIDI source. Standard MIDI messages are translated into audio unit parameter values. This interface is superseded by functions in the Music Player API.
- `DefaultAudioOutput.h`: Deprecated: Do not use. Defines an older interface for accessing the default output unit (deprecated in OS X v10.3 and later).

Figure 12

    2. AVFoundation Framework - Record, edit, and play photos, audio, and video; configure your audio session; and respond to changes in the device audio

environment. If necessary, customize the default system behavior that you implement with AVKit.



AVFoundation.framework

The AV Foundation framework provides an Objective-C interface for playing back audio with the control needed by most applications. The AV Foundation framework in iOS includes one header file:

- `AVAudioPlayer.h`: Defines an interface for playing audio from a file or from memory.

Figure 13

5. cJMAP APPLICATION

- The algorithm is developed for reducing the background noise and improving the speech quality and intelligibility in a noisy speech environment.
- For instance, if a hearing-impaired person is in a restaurant amidst many people blabbering in the background, which makes it difficult to understand the desired speech, they can connect their hearing aids to the smartphone with cJMAP app.
- Once the noise reduction in the app is turned on, the background noise reduces considerably, improving quality and intelligibility of the desired speech.
- The steps to be followed once the app is installed on your iOS device are as follows,
    a. Make sure you have the hearing aid device paired to your iPhone or iPad.
    b. For normal hearing people, wired headphones can be used instead of the hearing aids.
    c. Click on the icon by the name cJMAP, present amongst the apps.
    d. You will see a display as shown in Figure 9.

Figure 9

e. The noise reduction is initially turned off as shown in Figure 9, which plays back the original audio data from the environment. It is noisy data if you are in a noisy environment.

f. To turn ON noise reduction, just click on the button present on your screen and it will display ON indicating that noise reduction is working. Refer Figure 10.

Figure 10

g. **Note:** When you turn on noise reduction, the algorithm takes about 3 seconds to study the noise characteristics. It is advisable to turn on noise reduction (for the first time) when there is no speech, to achieve best performance.

h. A red slider in Figure 10 (referred as beta factor), is provided for user to control and strike a balance between noise reduction and speech distortion.

i. Initial value of slider is 0.5 at which point, the integrity of speech is well preserved, but noise is not suppressed satisfactorily. As you slide towards right (Figure 11), the noise reduces significantly. However, at some point, the speech is distorted

making the speech unintelligible.

Figure 11

j. The user can vary the slider based on their preference. When they are in an environment where there is no speech of interest (airports or inside flight), they can set the slider to have maximal noise reduction. When there is speech of interest, they can set the slider to a value that enables them to understand the speech with considerable noise reduction.

k. There is one more feature, which is provided to you in the app, i.e. to measure the Signal to Noise Ratio (SNR) at the microphone of the device. The SNR (in dB) will

be displayed by clicking on the press button below SNR as shown in Figure 12.

Figure 12

l. Please do repeat all the steps mentioned above every time you open the app and make sure to re-open the app and repeat the steps when you go to a different noisy environment.

m. In case the noise reduction is not satisfactory, we recommend you to restart the app and remain silent during noise training period of few seconds as soon as you start noise reduction.

- If you want to know more about the algorithm, do refer our paper, which is under review. All the codes will be made available in our website.