ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ (ΗΜΜΥ)

Ήχος σε πραγματικό χρόνο στο iOS

Real-Time Audio On iOS

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Mohammed Ragab

Βόλος, 25 Ιουνίου 2013

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών (ΗΜΜΥ)

Ήχος σε πραγματικό χρόνο στο iOS

Real-Time Audio On iOS

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Mohammed Ragab

Επιβλέποντες Καθηγητές:

Αλκιβιάδης Γ. Ακρίτας — Αναπληρωτής Καθηγητής

Δασκαλοπούλου Ασπασία — Επίκουρη Καθηγήτρια

Εγκρίθηκε από τη διμελή εξεταστική επιτροπή την 25η Ιουνίου 2013

(Υπογραφή)	(Υπογραφή)
.....	.....
Αλκιβιάδης Γ. Ακρίτας	Δασκαλοπούλου Ασπασία

Βόλος, 25 Ιουνίου 2013

(Υπογραφή)

.....

Mohammed Ragab

Διπλωματούχος του Τμήματος Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών (ΗΜΜΥ), Πανεπιστημίου Θεσσαλίας (ΠΘ).

Dedicated to my family

ΠΕΡΙΛΗΨΗ

Πώς θα σας φαινόταν το να μπορείτε να μετατρέψετε την οθόνη του κινητού σας ή του tablet σας σε ένα διαδρατικό μουσικό όργανο, όπως πιάνο, ντραμς ή ακόμα και ηλεκτρική κιθάρα; Ή τι θα λέγατε εάν το μικρόφωνο του κινητού σας την ώρα που μιλάτε σας άλλαζε τη φωνή σε πραγματικό χρόνο; Αυτά, χάρη στο Open Software Pure Data μπορούσαμε μέχρι τώρα να τα προγραμματίσουμε γραφικά στον υπολογιστή. Ωστόσο, η βιβλιοθήκη libpd είναι εκείνη που καθιστά εφικτή τη λειτουργία της εφαρμογής σε κινητά και tablets. Και ακόμη καλύτερα τι θα λέγατε εάν μπορούσατε να τα έχετε όλα αυτά δωρεάν στο κινητό/tablet σας;

Τόσο για έναν συνθέτη μουσικής που επιθυμεί να φτιάξει εφαρμογές ήχου σε πραγματικό χρόνο, όσο και για έναν προγραμματιστή που στοχεύει να βελτιώσει τις εφαρμογές του προσθέτοντας διαδραστικό ήχο, μέσω του συνδυασμός του Pure Data και της βιβλιοθήκης Libpd δίνεται η δυνατότητα για δημιουργία εφαρμογών ήχου σε πραγματικό χρόνο σε iOS περιβάλλοντα.

ABSTRACT

What if we could make any device or any software a re-programmable musical instrument, effect, or sound maker? Our phone could be a touch-controlled effect, our tablet a sketchpad for interactive drum sequencers. Patches assembled on our desk on a computer could be taken with us in our pocket. And what if we could do all of this for free, using a time-tested environment?

Libpd, authored by Peter Brinkmann, takes on that vision. It is a way of making Pure Data (or Pd), an open source of graphical programming environment for audio, and video processing in computer, the visual development tool for interactive music and media, more accessible across a range of applications and gadgets. It lets us embed Pd pretty much anywhere. It is not a new version of Pd. Instead, it makes use of the standard, “vanilla” distribution of the free and open source software. What is different is that it separates the sound processing part of Pd from the part that talks to audio hardware, allowing Pd to run on a greater variety of mobile devices and inside other applications.

Libpd:

- turns Pd into an audio synthesis and processing library
- liberates Pd from GUI and drivers
- allows for easy communication between Pd and the code into which it is embedded (so we can send and receive messages with our Pd patch)

Whether you are an audio developer looking to create applications with real-time audio, or you are an Applications developer ready to enhance mobile games with real-time audio, Pd and libpd can help you to create real-time audio applications for IPhone Operation System (iOS).

ACKNOWLEDGMENTS

The writing of this dissertation has been one of the most significant academic challenges I have ever had to face. Without the support, patience and guidance of the kind people around me this study would not have been completed.

My deepest of gratitude must go to my thesis advisor, Professor Akritas Alkiviadis. I have been amazingly fortunate to have an advisor who gave me the freedom to explore on my own and at the same time (real-time) the guidance to recover when my steps faltered. Professor Akritas taught me how to question thoughts and express ideas. His patience and support helped me overcome many crisis situations and finish this dissertation.

Special thanks to my committee, Professor Daskalopoulou Aspasia.

I would like to express my sincere gratitude to:

- ALL the stuff of University of Thessaly especially the department of Computer and Communication Engineering.
- Miller Puckette for creating and supporting the Pure Data open source.
- All libpd Team especially Peter Brinkmann, Dominik Hierner, and Martin Roth for building and developing the objective-C.
- Apple iOS developer Centre.

Finally, I wish to thank my family and friends, who have always believed in me and helped me to reach my goals.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF STEPS

1.INTRODUCTION

1.1. REAL-TIME AUDIO (RTA):

Real-Time Audio, is an application that was developed and written in Objective-C and lets us to test some MIDI notes by hearing the sound, with or without a delay. The users are able to speak and hear themselves in Real-time.

1.2. SUPPORTED PLATFORMS:

In each case, we just need libpd, Pd for making your patches (graphically), and a copy of the SDK for each mobile platform we want to use.

We can use libpd with:

IOS

We have tested the application RTA on our IPhone 4, iOS 6.1.3 and it worked in a high level of success.

Android

Thanks to Google's NDK (Native Development Kit), we can use libpd with any Android device running OS 1.6 or later.

1.3. About iOS App Programming

This document is the starting point for creating iOS apps. It describes the fundamental architecture of iOS apps, including how the code you write fits together with the code provided by iOS. This document also offers practical guidance to help you make better choices during your design and planning phase and guides you to the other documents in the iOS developer library that contain more detailed information about how to address a specific task.

1.4. THESIS STRUCTURE:

In this thesis we present the implementation of RTA, a simple Real-Time application. In Chapter 2 we mention the open software Pure Data (or Pd) Language features describing also the way it can be used. In Chapter 3 we display the importance of Libpd library as a means to run Pure Data not only in our computer but also in our smartphone and we explain the functions which allow us to embed the Pure Data. In Chapter 4 we provide the steps for the users in order to make their own application in a few minutes. Chapter 5 is referred to the Appendices (Other libraries for Real-Time Audio, Managing your Team, and finally more information about this App). Finally, Chapter 6 contains the bibliography.

1.5. REQUIREMENTS:

IOS device

Because of the fact that a simulator cannot receive voice messages, it is not the appropriate tester. Consequently, we will need an iOS device (iPhone) to test our application.

Git

Git comes from the Linux world and is so important in this application that we will need it for downloading and probably for updating your application.

Pure Data (Vanilla) Installed

We will need the program installed in our Mac and we have to be able to create and test some patches that we will use in our application.

2.PURE DATA (Pd) (Language features)

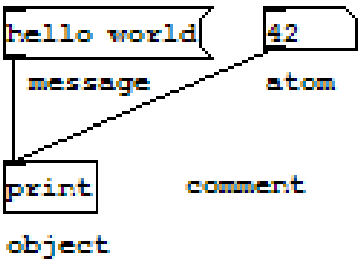


Figure 2.1 Hello World

*****Pd's four text objects: message, atom, object, and comments.

2.1. GENERAL INTRODUCTION:

Pure data (Pd) has been written and developed by Miller Puckette in 1990s, an open book for more tutorials, and includes the work of many developers.

It is a real-time graphical programming environment for audio, video and graphical processing that is used for live music performance, sound effects, composition, audio analysis, and interfacing with sensors, using cameras, controlling robots or even interacting with websites. These various media are handled as digital data within the program. (Holzer, 2013).

Pd has established itself as one of the leading open source packages for computer music and it remains largely interoperable with its commercial cousin Max/MSP. A pure data program is called a patch in a graphical representation of the flow of audio signals and control messages in a piece of music that Pd is executed in real time changes to a patch take effect immediately, its interactive and visual nature accounts for much of the appeal of pd. (puckette,1997).

Pd has been written in C and can be run on Linux, Windows, and Mac OS X, as well as mobile platforms like Android and iPhoneOS, using libpd.

2.2. PURE DATA (Pd) (Language features)

Pd was always designed to do control-rate of audio processing on the host Central Processing Unit (CPU), rather than offloading the sound synthesis and signal processing to a Digital Signal Processing (DSP). Similarly to Max, Pd is a data flow programming language. As with most (DSP) software, there are two primary rates at which data is passed, for example; (1) audio rate, usually at 44,100 samples per second, and (2) control rate, at 1 block per 64 samples. Control messages and audio signals generally flow from the top of the screen to the bottom between "objects" connected via inlets and outlets.

Pd is a great audio engine, it is powerful, flexible and extensible. With the appearance of libpd, Pd range has grown beyond the desktop to mobile and embedded setting. Its permissive of BSD license allows developers to add libpd to any project.

2.3. GRAPHICAL PROGRAMMING

Users of Pure Data can create new programs (patches) by placing functions (objects) on the screen. They can change the way these objects behave by sending them messages and by connecting them together in different ways by drawing lines between them.

2.4. REAL TIME

One of the most attractive features of Pd is that we can change the signal processing graph in the screen while Pd is running, and you can hear the results immediately. This makes it an excellent tool for prototyping audio components; Sound designers can prototype audio components in Pd using the same objects and techniques that they would use when creating a patch for any other purpose. When the patch is done the application developers simply add it to the resource of the application.

2.5. WHAT IS DIGITAL AUDIO?

Since we will be using Pure Data to create sound, and since Pd treats sound as just another set of numbers, it might be useful to review how digital audio works?

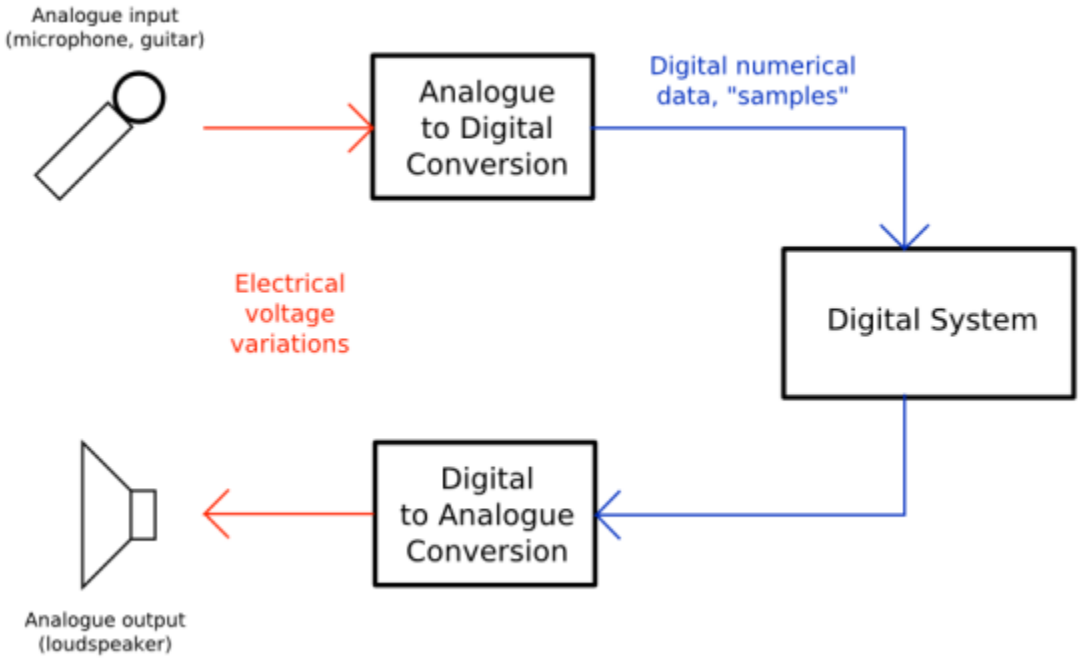


Figure 2.2 Digital Audio

2.6. PURE DATA (Pd Vanilla) MORE HELPS

Object Name	Function
Bang	Output a bang message
Trigger	Sequence and convert messages
Print	Print out message
Delay	Send a message after a time delay
Mtof	Convert MIDI to frequency
*~	Arithmetic on audio signals
Dac~	Audio output
Adc~	Audio input
Vline~	Generate audio ramps
Cos~	Cosine signal
Float	Store and recall number
Message object	Send messages to a named object
Number object	Send float numbers to a named object

Table 2.1 Pure Data more helps

2.7. MIDI

USING MIDI WITH PURE DATA

Figure 2.3 MIDI

Short for **M**usical **I**nstrument **D**igital **I**nterface is a great way to connect disparate pieces of hardware. MIDI technology was standardized in 1983 by a panel of music industry representatives, and was maintained by the MIDI Manufacturers Association. MIDI allows multiple instruments to be played from a single controller, which makes stage setups much more portable. This system fits into a single rack case, but prior to the advent of MIDI would have required four separate keyboard instruments, plus outboard mixing and effects.

In 2013, you can use a USB-MIDI interface which has been firstly used and connected to a synthesizer. From 1983 they will understand each other.

MIDI support in an application based on libpd which is usually redundant because of the fact that MIDI event will not encode any information that regular Pd messages cannot express.

A small table of MIDI numbers, frequencies, and their

Traditional names:

Frequenzy (Hz)	MIDI	Note name
65.4	36	C2
100	43	G2
130.8	48	C3
261.6	60	C4
277.1	61	C#4
293.6	62	D4
311.1	63	D#4
329.6	64	E4
349.2	65	F4
370	66	F#4
392	67	G4
415.3	68	G#4
440	69	A4
466.1	70	Bb4
493.8	71	B4
523.2	72	C5
1000	83	B5
4186	108	C8

Table 2.2 MIDI Note and Frequencies

3.Libpd

3.1. GENERAL INTRODUCTION:

Libpd itself is a thin wrapper on top of Pure Data that turns PD into a signal processing audio library. Through six core developers, many contributors can open Source (BSD License) that has been written in C, with bindings for java, Objective-C, C++, and Python.

3.2. WHO MADE Libpd?

Libpd was conceived by a team including Peter Brinkmann, Hans-Christoph Steiner, with input from the RjDj team (particulary Martin Roth). It was primarily developed by Peter Brinkmann, who applied his talents and the work he has done in JJACK, a Java API for JACK, with additional contributions and testing by Chris McCormick.

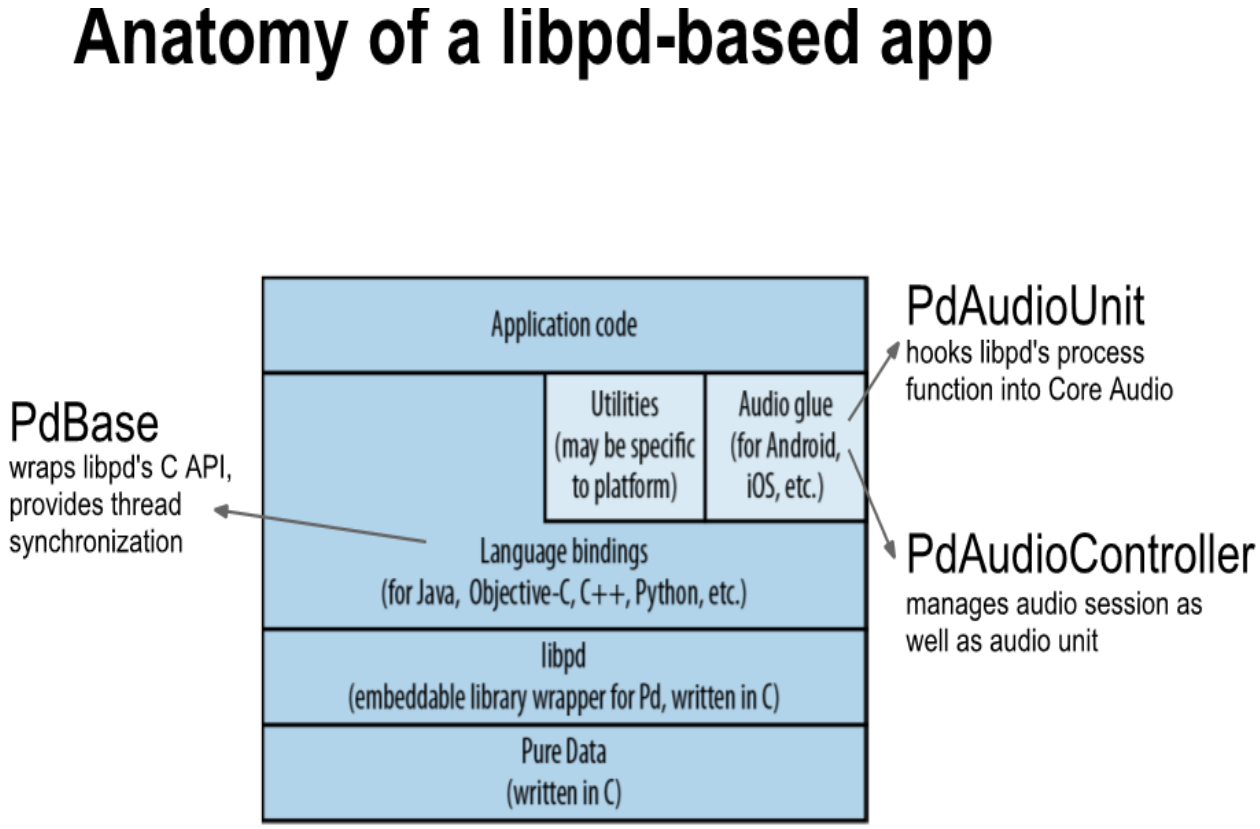


Figure 3.1 Libpd Anatomy

3.3. DOWNLOADING…

In order to download a copy of the iOS branch of libpd, we can open a terminal, change it into the directory where we want to keep our installation of libpd. Afterwards we must enter the following commands:

1. git clone git://github.com/libpd/pd-for-ios.git
2. cd pd-for-ios
3. git submodule init
4. git submodule update

To keep our copy of libpd up to date we must use the following commands:

5. git pull
6. git submodule update

Steps 3.1 download libpd

3.4. EMBEDDING PURE DATA WITH LIBPD:

Pd was originally designed to be an interactive tool for computer music and multimedia, tightly integrating a dataflow programming language.

3.4.1. Receiving Messages from Pd

We need to create a receiver (class) to receive messages (handling message) from Pd, then we register a variable (instance variable) with libpd, and finally we let libpd to know which send symbols in Pd we subscribe to. In order to receive messages from Pd, we register an instance of PdDispatcher with libpd. Finally in order to receive messages from send symbols in Pd, we need to implement a listener interface. Libpd will call them when a message from Pd arrives.

Creating a receiver (class) to receive messages (handling messages) from Pd requires firstly the registration of variables (instance variable) with libpd, and finally, libpd will recognize which symbols have been sent in Pd from those that we subscribed for.

3.4.2. Listener' s interface

Most of the listener's interface will need to handle all types of messages and therefore we are free to choose which method to implement.

Listener's interface for messages from Pd.

```
@protocol PdListener
@optional
- (void)receiveBangFromSource:(NSString *)source;
- (void)receiveFloat:(float)received fromSource:(NSString *)source;
- (void)receiveSymbol:(NSString *)symbol fromSource:(NSString *)source;
- (void)receiveList:(NSArray *)list fromSource:(NSString *)source;
- (void)receiveMessage:(NSString *)message withArguments:(NSArray
*)arguments fromSource:(NSString *)source;
@end
```

Receiver's interface for printing and receiving messages from Pd

```
@protocol PdReceiverDelegate<PdListener>
@optional
- (void)receivePrint:(NSString *)message;
@end
@interface PdBase {
  Not meant to be instantiated. No member variables.
}
+ (void)initialize;
+ (size_t)setMessageBufferSize:(size_t)size;
+ (void)setDelegate:(NSObject<PdReceiverDelegate> *)newDelegate;
```

PdBase retains the delegate: call setDelegate with nil in order to release delegate.

```
+ (NSObject<PdReceiverDelegate> *)delegate;
```

3.4.3. Setting the search path

Furthermore, we can add resources, such as a wave files.

```
+ (void)clearSearchPath;
+ (void)addToSearchPath:(NSString *)path;
```

3.4.4. Opening Patches

We can open a patch by sending the path of the patch to libpd, and get back a handle that identifies the patch. We could also open multiple copies. A handle is simply a pointer for the data structure that represents the patch in Pd.
(Libpd will initialize by PdBase automatically).

```
+ (void *)openFile:(NSString *)baseName path:(NSString *)pathName;
+ (void)closeFile:(void *)x;
+ (int)dollarZeroForFile:(void *)x;
```

3.4.5. Sending Messages to Pd

Pure Data supports different message types such as bang, float, symbol, list, and typecast string messages with arguments. There is a method that sends a message to Pd which receives the symbols. All the message types must be string or numbers.

At this point, we should note that the pointer means nothing outside of Pd. If we want to support pointers in Pd, we can store them in a pointer object in our patch and then trigger them with a bang from libpd.

+ (int)sendBangToReceiver:(NSString *)receiverName;

+ (int)sendFloat:(float)value toReceiver:(NSString *)receiverName;

+ (int)sendSymbol:(NSString *)symbol toReceiver:(NSString *)receiverName;

+ (int)sendList:(NSArray *)list toReceiver:(NSString *)receiverName; // list may be nil

+ (int)sendMessage:(NSString *)message withArguments:(NSArray *)list toReceiver:(NSString *)receiverName; // list may be nil

3.4.6. Arrays

In Pd, arrays are very useful for storing many samples or calculating function values, and the user' s interface of Pd will let you view and modify arrays in real-time.

Reading and Writing arrays in Pd

Accessing Pd arrays

+ (int)arraySizeForArrayNamed:(NSString *)arrayName;

+ (int)copyArrayNamed:(NSString *)arrayName withOffset:(int)offset toArray:(float *)destinationArray count:(int)n;

+ (int)copyArray:(float *)sourceArray toArrayNamed:(NSString *)arrayName withOffset:(int)offset count:(int)n;

3.5. MIDI SUPORT IN LIBPD

Sending MIDI messages

All the parameters are integers and their value ranges from 0 to 127.

However, we have an exception for channel numbers and pitch bend values.

+ (int)sendNoteOn:(int)channel pitch:(int)pitch velocity:(int)velocity;

+ (int)sendControlChange:(int)channel controller:(int)controller value:(int)value;

+ (int)sendProgramChange:(int)channel value:(int)value;

+ (int)sendPitchBend:(int)channel value:(int)value;

+ (int)sendAftertouch:(int)channel value:(int)value;

+ (int)sendPolyAftertouch:(int)channel pitch:(int)pitch value:(int)value;

+ (int)sendMidiByte:(int)port byte:(int)byte;

+ (int)sendSysex:(int)port byte:(int)byte;

+ (int)sendSysRealTime:(int)port byte:(int)byte;

4.My Application

Figure 4.1 My App

4.1. GENERAL INTRODUCTION

In this Chapter we will quickly present the most important steps that we will definitely need to make our own application in a few minutes.

Most applications should stick to the following sequence:

- Import libpd
- Initialize the audio component.
- Create a dispatcher and register it with PdBase.
- Load your patches.
- Send and Receive Massages
- Start the audio components.
- Close all patches and release the dispatcher

Steps 4.1 make your app

4.1.1. Import libpd

In order to use libpd in an app, we need to import the libpd Xcode project into our project.

Click on the menu item ➔ File ➔ add Files to “your app” and add the file libpd.xcodeproj from the directory you downloaded the libpd files.

- Build Settings tab

Enter in the search term “user header” and find the labels “User Header Search Paths” and add the path: $(SRCROOT)/…/…/pd-dor-ios/libpd/objc.

- Build Phases tab

Target Dependencies section

✓ Add libpd-ios

Link Binaries with Libraries section add:

✓ Libpd-ios-a
✓ AudioToolbox.framework
✓ AVFoundation.framework

*make sure to select “Create group for any added folders.

4.1.2. Initialize the audio component.

The most important rule is that we should initialize the audio components before we open any patch, because some patches will query Pure Data for audio properties like the sample rate upon loading. We need to request input channels when configuring the audio components.

```
PdAudioController audioController = [[PdAudioController alloc] init];

[self.audioController configurePlaybackWithSampleRate:44100
numberChannels:2 inputEnabled:YES mixingEnabled:NO];
```

4.1.3. Create a dispatcher and register it with PdBase

It is a good idea to register a dispatcher early on, even if we do not intend to add any listeners, because it will log console message from Pure Data that may give us useful debugging. We can add or remove listeners at any time, but if we open patches before, we may miss messages from Pure Data.

```
PdDispatcher dispatcher = [[PdDispatcher alloc] init];
[PdBase setDelegate:dispatcher];
```

4.1.4. Load patches.

Let's add a Pure Data to our app. In order to add patches we need a handler or a pointer to an internal data structure representing the patch. We can open multiple instances of each patch.

```
Void *myNotesPatch = [PdBase openFile:@"myPd.pd" path:[[NSBundle
mainBundle] resourcePath]];
Void *myAudioPatch = [PdBase openFile:@"myAudioPatch.pd"
path:[[NSBundle mainBundle] resourcePath]];
```

4.1.5. Send and Receive Messages

In order to send messages to Pure Data patches we have to control our patches. For receiving information out of Pure Data with a class that implements the Pd Receiver Delegate protocol.

```
[PdBase sendFloat : Float toReceiver:@"delay"];
[PdBase sendFloat: Float toReceiver:@"midinote"];
[PdBase sendBangToReceiver:@"bang"];
```

4.1.6. Start the audio components.

In order to start audio we just need to active the audioController.

```
PdAudioController audioController.active = YES;
```

4.1.7. Close all patches and release the dispatcher

When done do not forget to close patches and release the dispatcher.

```
[PdBase closeFile:myAudioPatch];
[PdBase closeFile:myNotesPatch];
[PdBase setDelegate:nil];
```

5. Appendices

5. A. OTHER LIBRARIES FOR REAL-TIME AUDIO

5.A.1. Mobile Music Toolkit (MoMu)

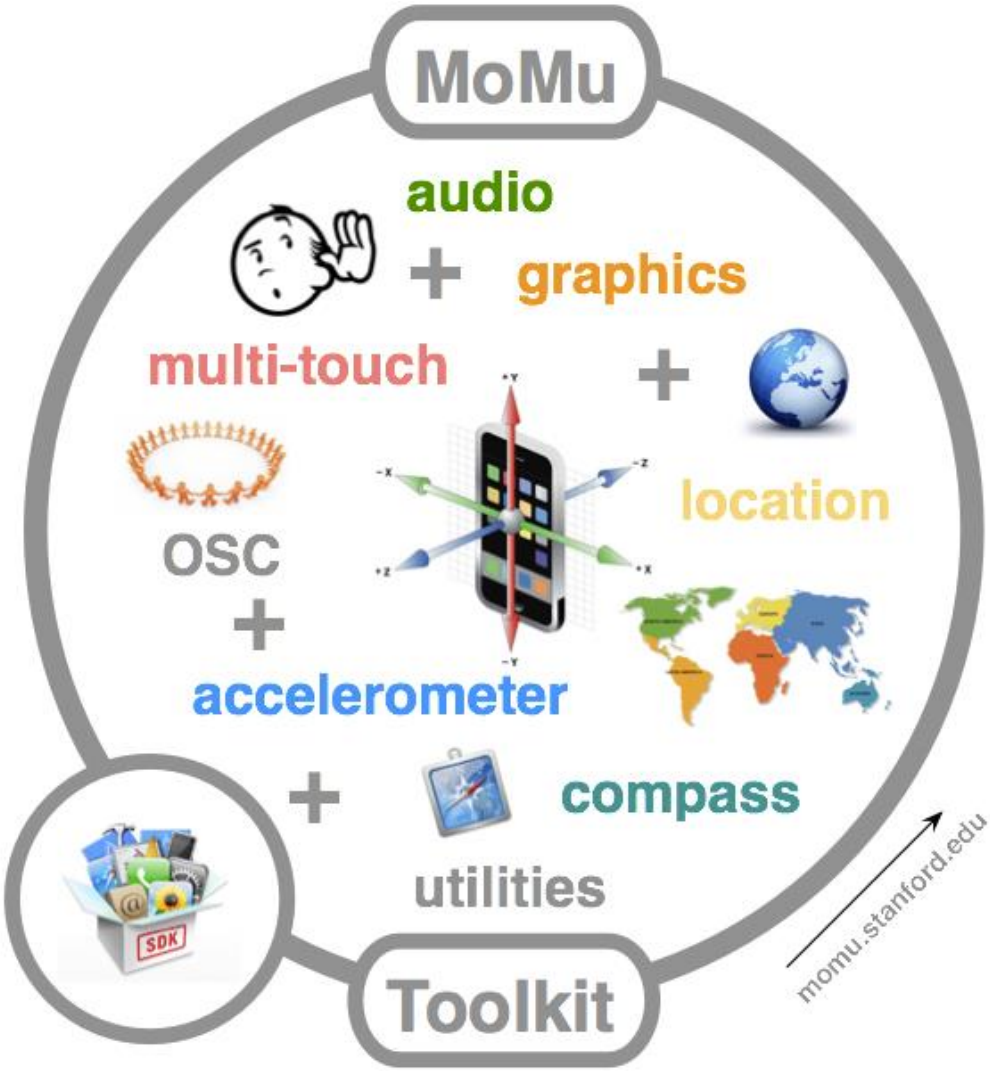


Figure 5.1 mobile music (MoMu)

MoMu is a light-weight software toolkit for creating musical instruments and experiences on mobile device, and currently supports the iPhone platform (iPhone, iPad, iPod Touches). MoMu provides API's for real-time full-duplex audio, accelerometer, location, multi-touch, networking (via Open Sound Control) , graphics, and utilities. The MoMu Toolkit was developed as part of the Mobile Music research initiative in Music, Computing & Design Group at Stanford University's CCRMA, in collaboration with Smule. MoMu is released under a BSD-like license.

5.A.2 Core Audio

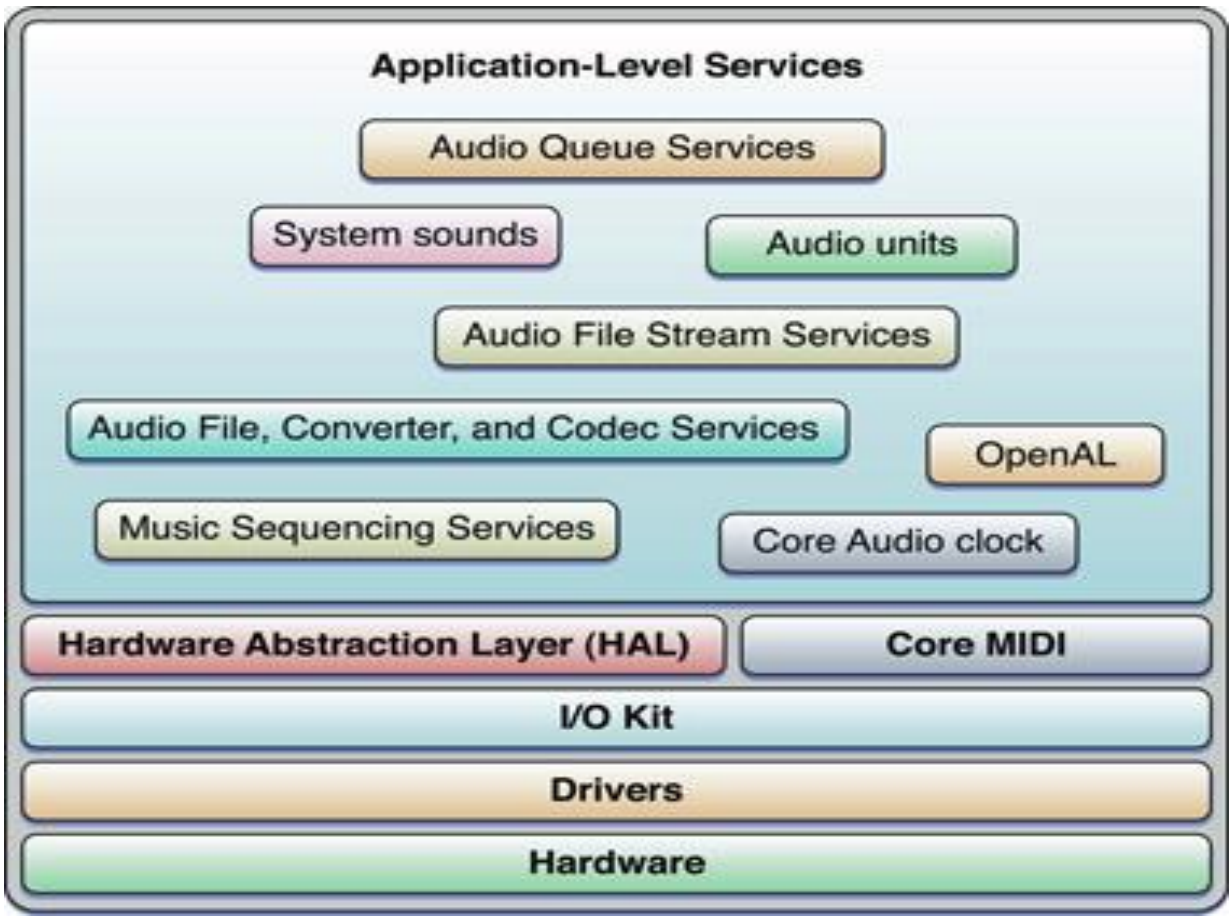


Figure 5.2 Core Audio

Core Audio is a low-level API for dealing with sound in Apple's Mac OS X and iOS operating systems. It includes an implementation of the cross-platform OpenAL library.

Apple's Core Audio documentation states that "in creating this new architecture on Mac OS X, Apple's objective in the audio space has been twofold. The primary goal is to deliver a high-quality, superior audio experience for Macintosh users. The second objective reflects a shift in emphasis from developers having to establish their own audio and MIDI protocols in their applications to Apple moving ahead to assume responsibility for these services on the Macintosh platform."

5. B MANAGING YOUR TEAM

After the team agent has joined a developer program, he adds other people to the team and sets their privileges. If you are the team agent and the sole developer on your team, no additional configuration is needed, because the team agent always has access to all account features.

5.B.1. Understanding Membership Privilege Levels

Lists the roles a team participant can play and provides a basic description of each. Each level of access includes all the capabilities of the levels below it.

Role	Description
Team agent	A team agent is legally responsible for the team and acts as the primary contact with Apple. The team agent can change the access level of any other member of the team.
Team admin	A team admin can set the privilege levels of other participants, although a team Admin cannot demote the team agent. Team admins manage all assets used to sign your apps, either during development or when your team is ready to distribute an app. Team admins are the only people on a team that can sign apps for distribution On no development devices. Team admins also approve signing certificate requests Made by team members.
Team Member	A team member gains access to pre-release content delivered by Apple on that Program's portal. A team member can also sign apps during development, and but only after he or she makes a request for a development signing certificate and has That request approved by a team admin.

Table 5.1 roles of Team Members

Privileges assigned to each membership level:

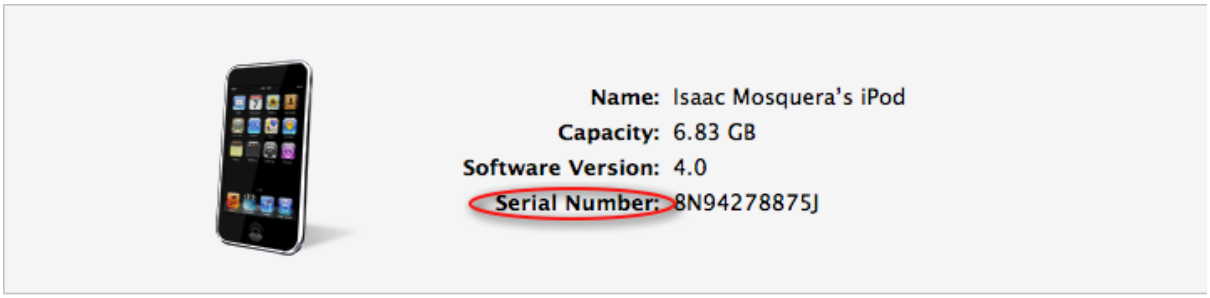
Privilege	Team agent	Team admin	Team member
Legal responsibility for the team	✓	✗	✗
Primary contact with Apple	✓	✗	✗
Invite team admins and team members	✓	✓	✗
Approve a request for a development signing certificate	✓	✓	✗
Add devices for development and user testing	✓	✓	✗
Create app IDs	✓	✓	✗
Request a distribution signing certificate from Apple	✓	✓	✗
Create development and distribution provisioning profiles	✓	✓	✗
Enable app IDs to use Apple Push Notification or In-App Purchase	✓	✓	✗
Create SSL certificates for the Apple Push Notification Service	✓	✓	✗
Request a development signing certificate	✓	✓	✓
Download development provisioning profiles	✓	✓	✓
View prerelease website content	✓	✓	✓

Table 5.2 Privileges assigned to each membership level

5.B.2. For Members:

To be able to test apps on your device, you will need to first have an Apple Developer Account, and then add your device to that account using its UDID.

You can retrieve your UDID by syncing your device with your iTunes account, and clicking on "Serial Number" from the within the Summary tab:



After doing so, you'll notice that this line now displays your "Identifier (UDID)":

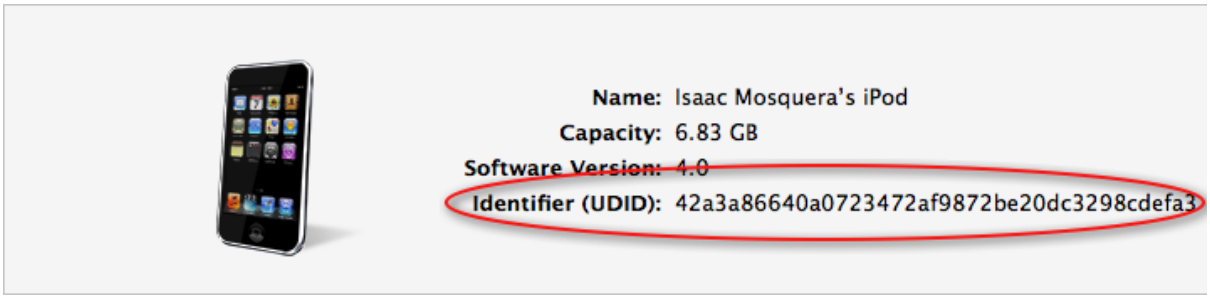
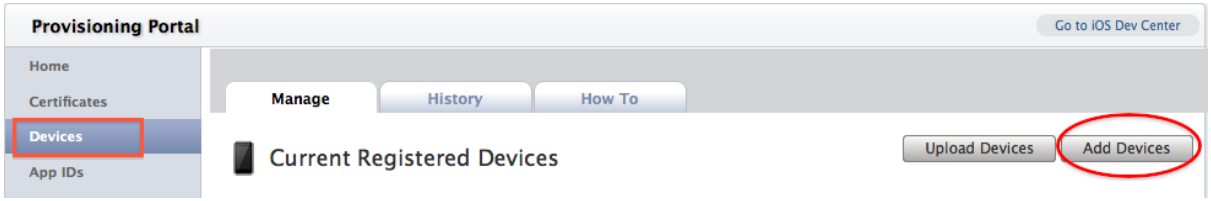


Figure 5.3 IPhone UDID

Now that you know your UDID, send an E-mail to your Admin/Agent team with your E-mail developer and UDID to Add your device. Then you will be able to download and install your Provisioning Profile.

Agent or Admin Member

Now that you have the UDID, go to the Devices section of your iOS Provisioning Portal and click "Add Devices":



Enter in your Device Name and UDID (it's best to copy and paste) and hit Submit:

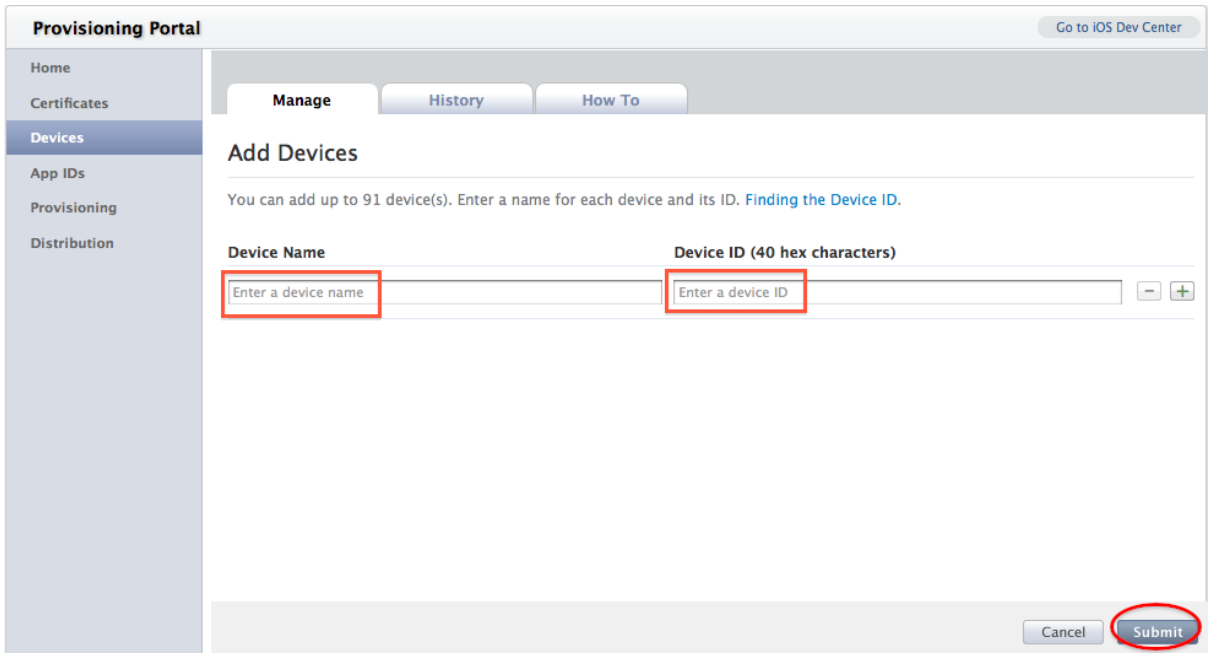


Figure 5.4 Add Devices

After you done you have to add an invitation to your members.

5.B.3. Adding Team Admins and Members

You can add people to your development team through the Member Centre. When you add a person to your team, you can grant his/her accesses to the developer programs that your team is enrolled in.

Steps

1. After logging in to the Member Centre, click People in the bar at the top.
2. Click Invitations in the sidebar.
3. Click Invite Person and provide the first name, last name, and email address.
4. Specify the person's access and role for each program.
5. Click Send Invitation.
6. Approve

Steps 5.1 Add Members

* The person you specified will receive an email of invitation, which he must verify by clicking on the invitation code. If the person does not have an Apple ID, he will be asked to create one before accepting the invitation.

5.B.4. To approve a development certificate request

1. In Certificates, Identifiers & Profiles, select Certificates.
2. Under Certificates, select Pending.
3. Select the certificate.
4. Click either Decline or Approve.

Steps 5.2 Approve a development certificate request

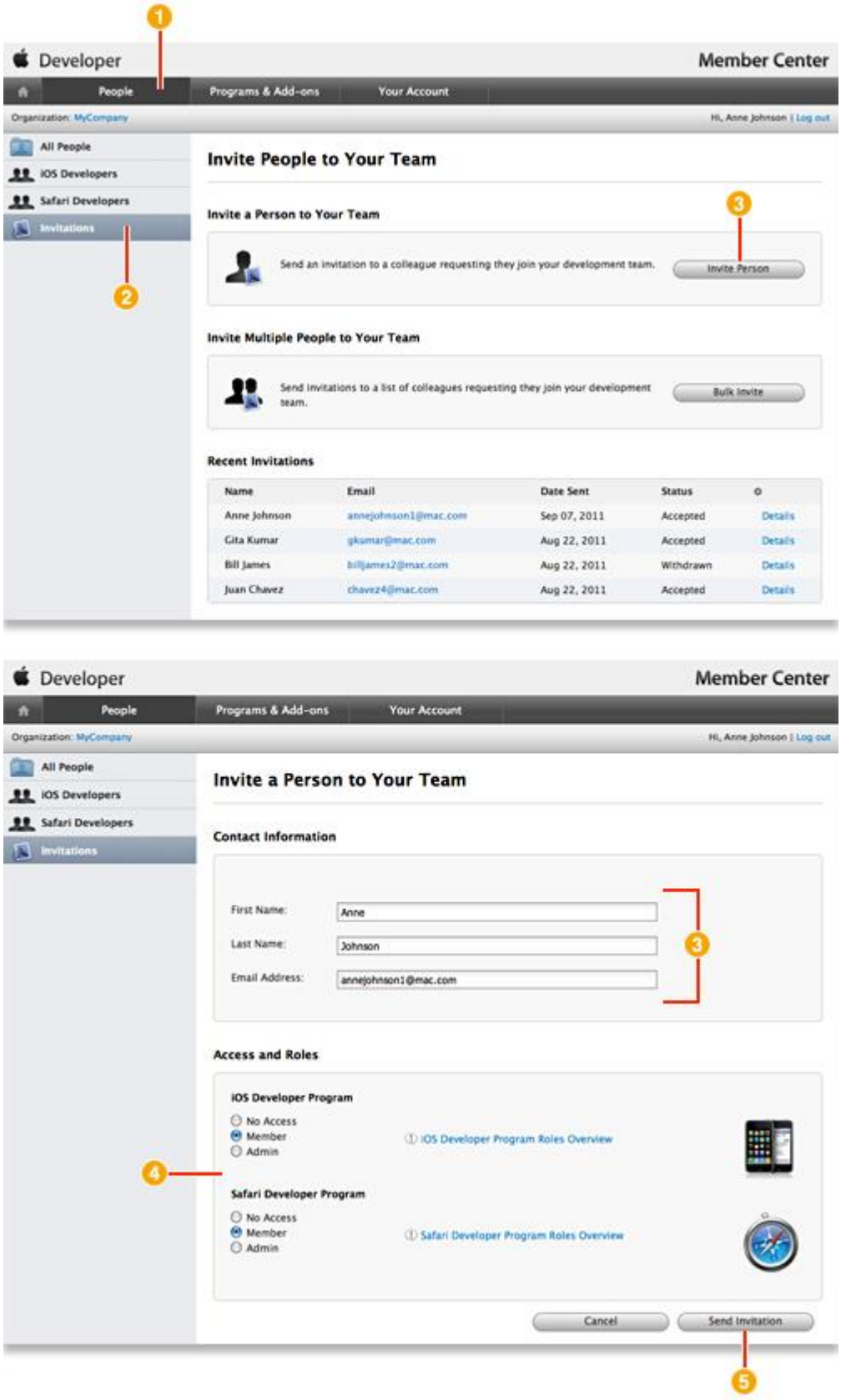


Figure 5.5 Add Members

Approving Development Certificates

Team members need a development certificate in order to sign apps, to use the team provisioning profile, or to be added to other provisioning profiles. Team admins are notified via email when a team member requests a development certificate. The email contains a link to Member Center to approve the request.

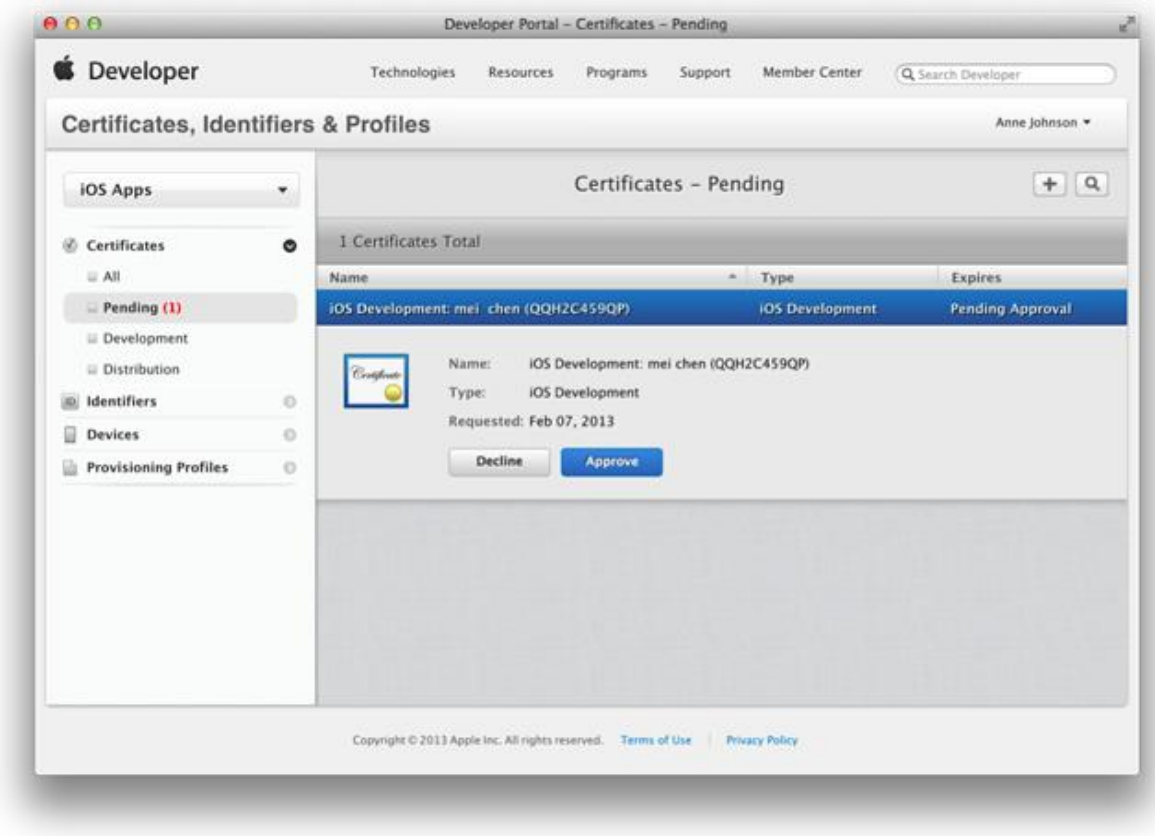


Figure 5.6 Approving Development Certificates

5.B.5. Editing a Team Member's Privileges

As your team grows, you may need to edit a team member's privileges. By changing a person's role in the Member Centre, the team agent or a team admin can grant that person more or fewer privileges.

Steps

7. After logging in to the Member Centre, click People in the bar at the top.
8. Click All People in the sidebar.
9. Click Details in the last column in the row of the person whose privileges you want to change.
10. Specify the person's access and role for each program and click Save.

Steps 5.3 Editing Members

Editing a Team Member's Privileges

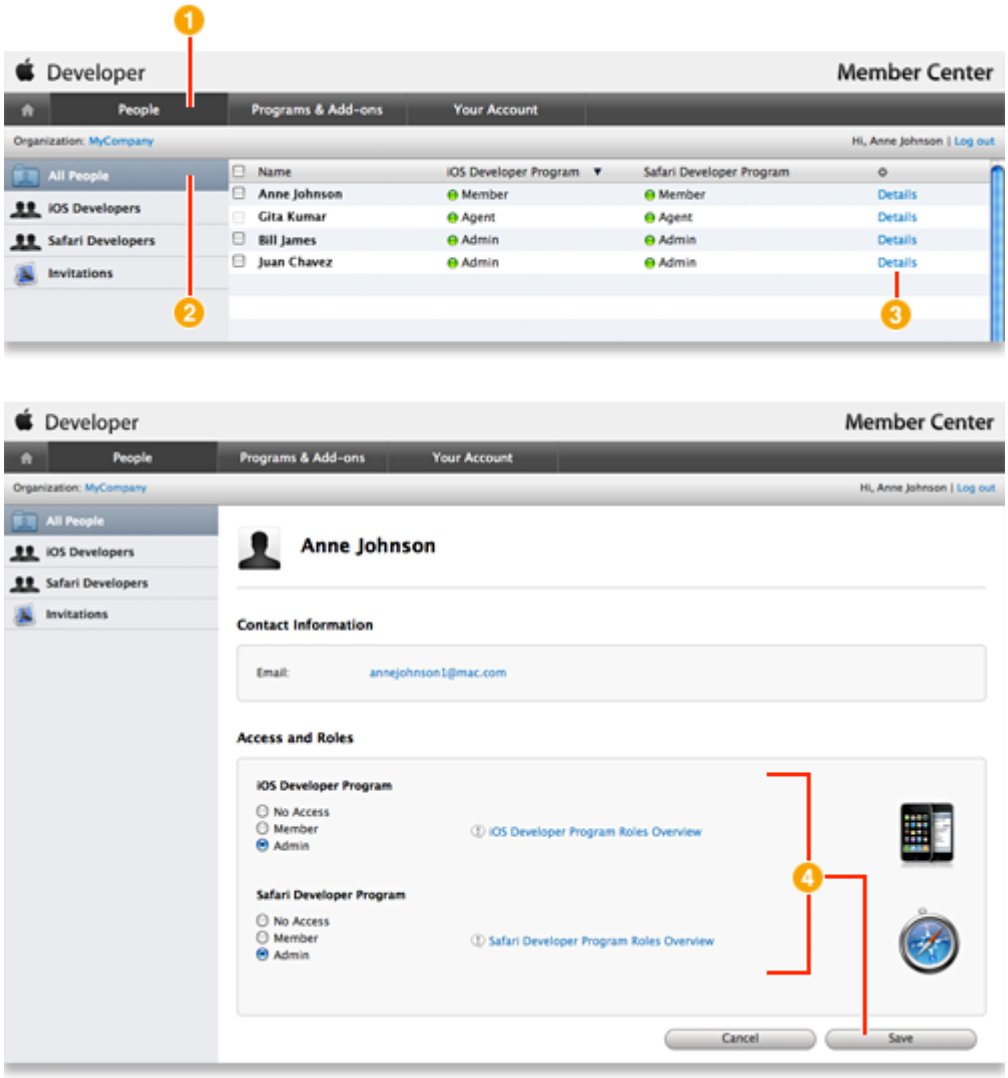


figure 5.7 Editing Members

5.C

Figure 5.8 iOS 7 Home Screen

iOS (previously iPhone OS) is a mobile operating system developed and distributed by Apple Inc. Originally unveiled in 2007 for the iPhone, it has been extended to support other Apple devices such as the iPod Touch (September 2007), iPad (January 2010) and second-generation Apple TV (September 2010). Unlike Microsoft's Windows Phone and Google's Android, Apple does not

license iOS for installation on non-Apple hardware. As of June 2013, Apple's App Store contained more than 900,000 iOS applications, 375,000 of which were optimised for iPad. These apps have collectively been downloaded more than 50 billion times. It had a 21% share of the smartphone mobile operating system units shipped in the fourth quarter of 2012, behind only Google's Android. In June 2012, it accounted for 65% of mobile web data consumption (including use on both the iPod Touch and the iPad). At the half of 2012, there were 410 million devices activated. According to the special media event held by Apple on September 12, 2012, 400 million devices have been sold through June 2012.

5.C.1 IOS devices

Figure 5.9 iOS Devices

IOS devices were designed and marketed by Apple Inc.(formerly. Apple Computer Inc.) That run a Unix-like operating system named iOS, often referred to simply as iDevices. The devices include the iPhone multimedia smartphone, the iPod Touch, which is similar to the iPhone but has no cell phone hardware, and the iPad tablet computer. All three devices function as audio and portable media players and Internet clients. The Apple TV, which ran iOS from the second generation of hardware onwards, is a set-top box for streaming local media to a connected television set, and has no screen of its own.

The operating system on iOS devices can be updated through iTunes, or, on iOS 5 or later, using firmware-over-the-air (FOTA). A major version of iOS tends to be released every time a new type of iPhone is launched, (about once a year) and is normally free, although iPod Touch users were formerly required to pay for the update. Apple upgrades its products' hardware periodically (approximately yearly). There have been six generations of iPhone (original iPhone, iPhone 3G, iPhone 3GS, iPhone 4, iPhone 4S and iPhone 5), five of iPod Touch (1st to 5th generations), and four of iPad (iPad (1st generation), iPad 2, iPad (3rd generation)and iPad (4th generation)).

5.C.2 IOS – Xcode

Installation

1. Download Xcode latest version
from (https://developer.apple.com/downloads/)

2. Double click the Xcode dmg file.

3. You will find a device mounted and opened.

4. Here there will be two items in the window that's displayed namely Xcode application and the Application folder's shortcut.

5. Drag the Xcode to application and it will be copied to your applications.

6. Now Xcode will be available part of other applications from which you can select and run.

Steps 5.3 Xcode download

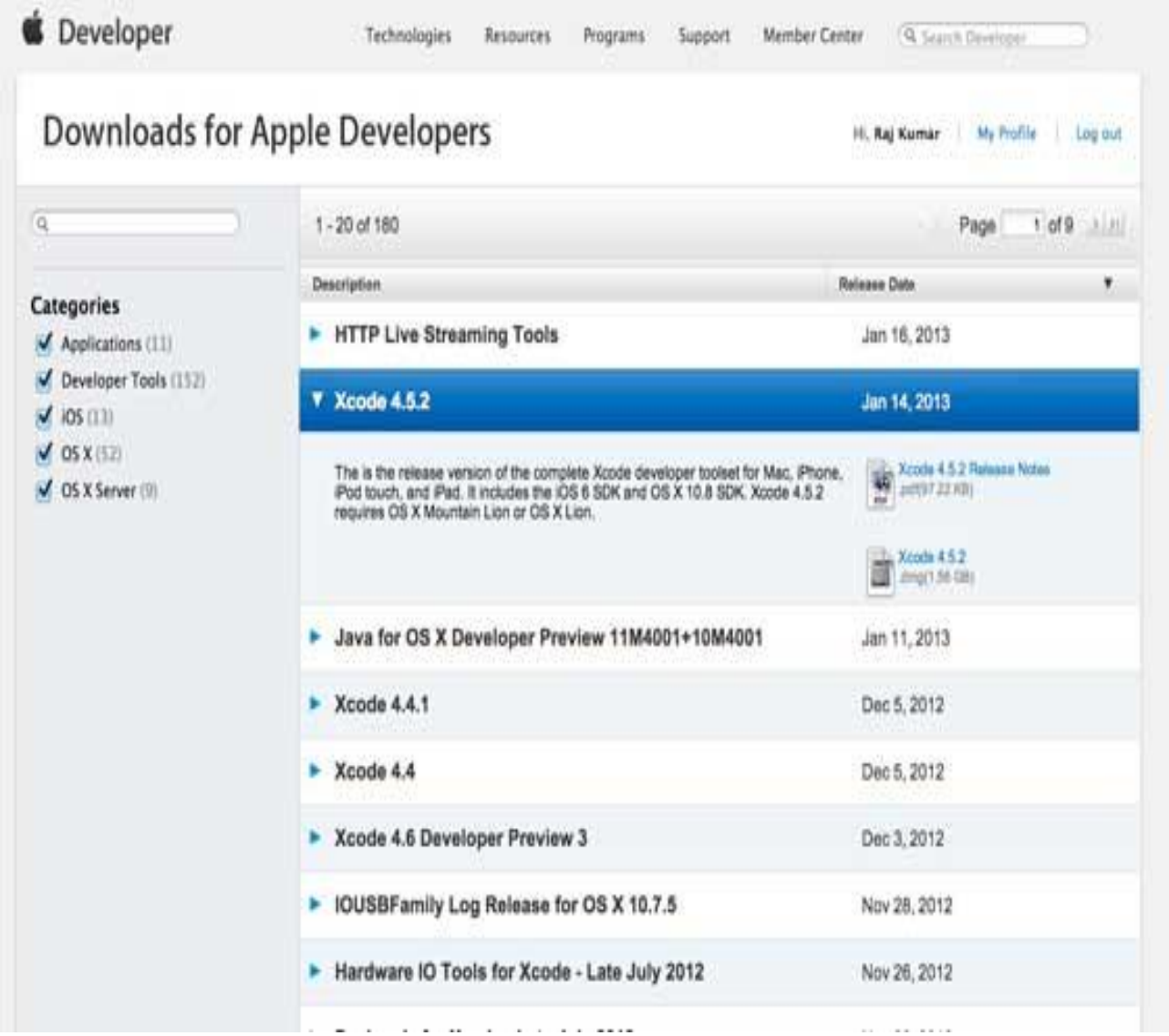


Figure 5.10 Xcode Download

You also have another option of downloading Xcode from the Mac App store and then install following the step by step procedure given in the screen.

5.C.3 Interface Builder

Interface builder is the tool that enables easy creation of UI interface. You have a rich set of UI elements that is developed for use. You have just drag and drop into your UI view.

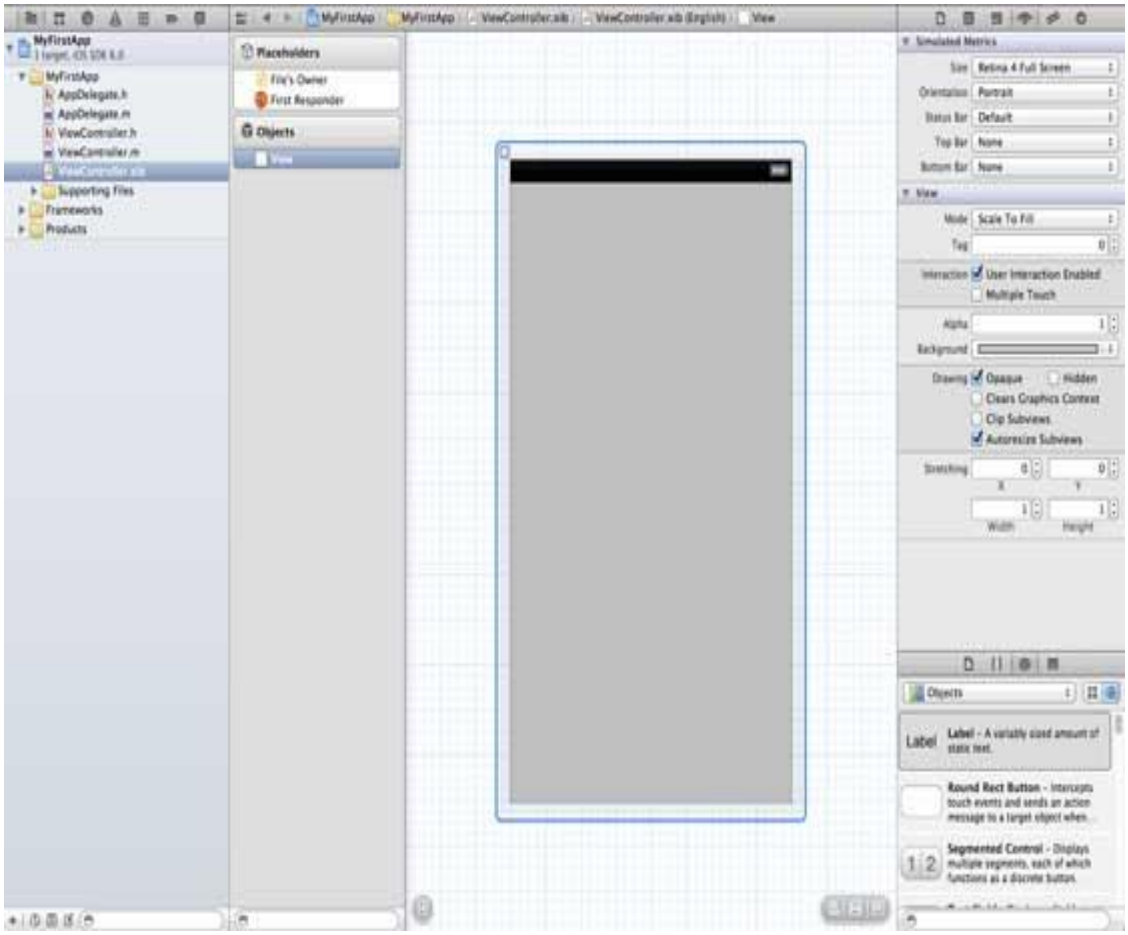


Figure 5.11 Xcode interface Builder

You have objects library in right bottom that consists the entire necessary UI element. The user interface is often referred as xibs which is their file extension. Each of the xibs is linked to a corresponding view controller.

5.C.4 iOS simulator

iOS simulator actually consists of two types of devices, namely iPhone and iPad with its different versions. iPhone versions include iPhone (normal), iPhone Retina, iPhone 5. iPad has iPad and iPad Retina

Figure5.12 iPhone Simulator

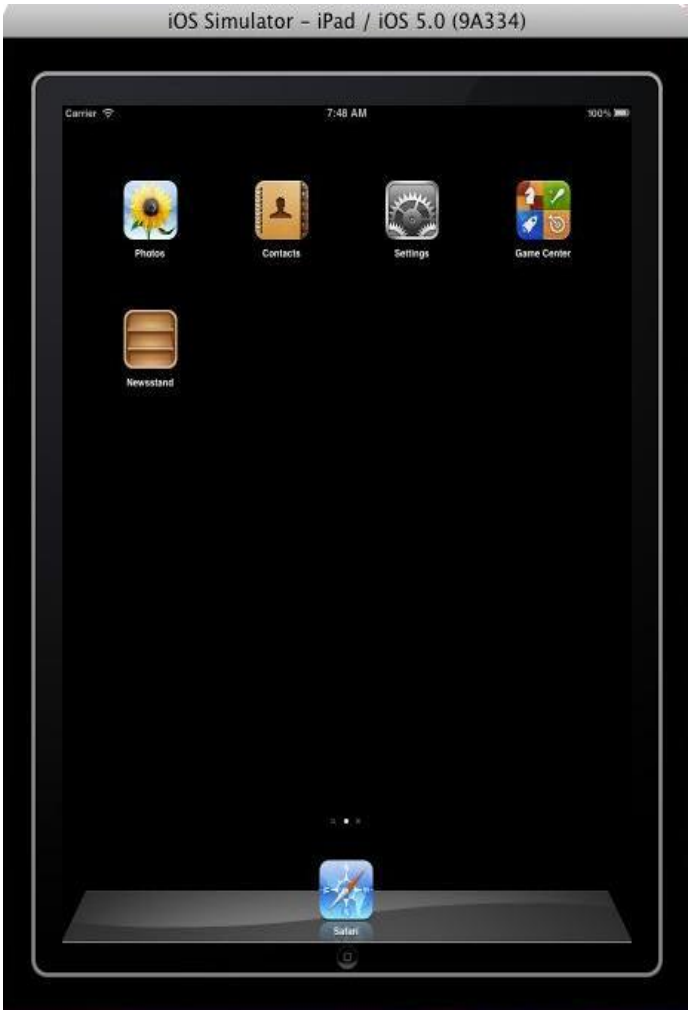


figure 5.13 iPad Simulator

We can simulate location in iOS simulator for playing around with latitude and longitude effects of the app. We can also simulate memory warning and in-call status in the simulator. We can be able to use the simulator for most purposes. But we cannot test the device features like accelerometer. So we might always need an iOS device to test thoroughly on all aspect and scenarios of an application.

5.C.5 **Objective C**

Introduction

The language used in iOS development is objective C. It is an object oriented language and hence it would easy for those who have some background in object oriented language programming.

Interface and Implementation

In objective C the file where the declaration of class is done is called the interface file and the file where the class is defined is called the implementation file.

A simple interface file MyClass.h .

```
@interace MyClass:NSObject{
// class variable declared here
}
// class properties declared here
// class methods and instance methods declared here
@end
```

The implementation file MyClass.m

```
@implementation MyClass
// class methods defined here
@end
```

Object Creation

Object creation is done as

```
MyClass  *objectName = [[MyClass alloc]init] ;
```

Methods

Class method:

Class methods can be accessed directly without creating objects for the class. They don't have any variables and objects associated with it.

```
+(void)simpleClassMethod;
```

It can be accessed by using the class name (let's assume the class name as MyClass) as follows.

```
[MyClass simpleClassMethod];
```

Instance methods:

Instance methods can be accessed only after creating an object for the class. Memory is allocated to the instance variables

```
-(void)simpleInstanceMethod;
```

It can be accessed after creating an object for the class as follows

```
MyClass  *objectName = [[MyClass alloc]init] ;
[objectName simpleInstanceMethod];
```

Important data types in Objective C

S.N.	*Data Type*
1	*NSString* It is used for representing a string
2	*CGfloat* It is used for representing a floating point value (normal float is also allowed but it's better to use CGfloat)
3	*NSInteger* It is used for representing integer
4	*BOOL* used for representing Boolean(YES or NO are BOOL types allowed)

Table 5.3 Data Type in Objective-C

Printing logs

NSLog - used for printing a statement. It will be printed in device logs and debug console in release and debug modes respectively.

NSlog(@"Testing my NSlog");

Control Structures

Most of control structures are same as in C and C++ except for a few additions like for in statement.

Properties

For an external class to access class variables properties are used

@property(nonatomic , strong) NSString *myString;

Accessing Properties

We can use dot operator to access properties.

```
self.myString = @"Test";
```

We can also use set method as follows.

```
[self setMyString:@"Test"];
```

Categories

Categories are used to add methods to existing classes. By this way we can add method to classes for which we don't have even implementation files where the actual class is defined.

```
@interace MyClass(customAdditions)
- (void)sampleCategoryMethod;
@end

@implementation MyClass(categoryAdditions)

-(void)sampleCategoryMethod{
  NSLog(@"Just a test category");
}
```

Arrays

NSMutableArray and NSArray are the array classes used in objective C. As the name suggests the former is mutable and latter is immutable.

```
NSMutableArray *aMutableArray = [[NSMutableArray alloc]init];
[anArray addObject:@"firstobject"];
NSArray *aImmutableArray = [[NSArray alloc]
initWithObjects:@"firstObject",nil];
```

Dictionary

NSMutableDictionary and NSDictionary is the dictionary classes used in objective C. As the name suggests the former is mutable and latter is immutable.

```
NSMutableDictionary*aMutableDictionary = [[NSMutableArray alloc]init];
[aMutableDictionary setObject:@"firstobject" forKey:@"aKey"];
NSDictionary*aImmutableDictionary= [[NSDictionary
alloc]initWithObjects:[NSArray arrayWithObjects:
@"firstObject",nil] forKeys:[ NSArray arrayWithObjects:@"aKey"]];
```

5.C.6  Cocoa Touch

Cocoa Touch is the programming framework driving user interaction on iOS. Using technology derived from Cocoa and the gorgeous Mac desktop, Cocoa Touch and the iOS interface were completely re-designed for multi-touch. Buttons, table lists, page transitions, and gestures on the iPhone are unique for the pocket able form factor, and all this UI power is available to developers through the Cocoa Touch frameworks.

Built upon the Model-View-Controller paradigm, Cocoa Touch provides a solid foundation for creating state-of-the-art applications. When combined with the Interface Builder developer tool, it is both easy and fun to use drag-and-drop to design the next great iOS application.

Strong low-level foundations enable fantastic high-level frameworks such as Game Kit for multiplayer gaming, Core Data, which offers high performance, yet easy-to-use data management, Core Animation for stunning effects, and the most innovative browser engine on mobile devices in WebKit.

Working together, the Cocoa Touch frameworks and powerful foundation provide a truly unique canvas upon which to create a new work of application art.

5.D.1 Real-Time Audio (RTA)

Who made RTA?

This application was designed and developed By Mohammed Ragab, Undergraduate student at department of Electrical and Computer Engineering (ECE), University of Thessaly, Volos, Greece. Supervisor teacher is Alkiviadis Akritas professor at University of Thessaly.

How does RTA work?

Headphone Required.

Figure 5.14 Real-Time Audio Interface

Piano Screen (have fun) :

Just play and enjoy your new little Piano in your phone.

Figure 5.15 Piano Interface

6.Bibliography

[1] IOS 5 Programming Cookbook by Vandad Nahavandipoor. Revision History http://oreilly.com/catalog/errata.csp?isbn=9781449311438.

[2]Apple iOS Developer Centre https://developer.apple.com/devcenter/ios/index.action .

[3] Miller Puckette main home Page http://crca.ucsd.edu/~msp/.

[4] Pure Data online tutorial English Book http://www.pd-tutorial.com/english/index.html.

[5] Pure Data Main Home Page for More Information's And Downloading the Program http://puredata.info/.

[6] Libpd Main Home Page, http://libpd.cc/documentation/.

[7] Libpd Main Source GitHub. https://github.com/libpd.

[8] Apple Main Core Audio Library Documentation, http://developer.apple.com/library/ios/#documentation/MusicAudio/Conceptual/CoreAudioOverview/Introduction/Introduction.html.

[9] Stanford Main Mobile Music Toolkit (MoMu) Documentation, http://momu.stanford.edu/.

[10] Pure Data, From Wikipedia, the free encyclopaedia,

http://en.wikipedia.org/wiki/Pure_Data

[11] MIDI, From Wikipedia, the free encyclopaedia,

http://en.wikipedia.org/wiki/MIDI.

[12] List of ios device
https://en.wikipedia.org/wiki/List_of_iOS_devices

[13] IOS, From Wikipedia, the free encyclopaedia,
http://en.wikipedia.org/wiki/IOS

[14] Objective-C, From Wikipedia, the free encyclopaedia,
http://en.wikipedia.org/wiki/Objective-C

[15] Core Audio, From Wikipedia, the free encyclopaedia,
http://en.wikipedia.org/wiki/Core_Audio

[16] Digital Audio http://en.flossmanuals.net/ardour/ch003_what-is-digital-audio/_booki/ardour/static/PureData-WhatIsDigitalAudio-Analogue_Digital_Conversion-en.png