Game Kit

iPhone and iPod touch Development
Fall 2009 — Lecture 22

Questions?

Announcements

- Assignment #8 (the last of the week-long assignments) out this past Saturday
  - Due next Monday

Today's Topics

- Game Kit Introduction
- Sessions and Peer Pickers
- Chat Demo
- In-Game Voice
- Voice Chat Demo

Notes

- I'm showing the relevant portions of the view controller interfaces and implementations in these notes
- Remember to release relevant memory in the -dealloc methods — they are not shown here
- You will also need to wire up outlets and actions in IB
- Where delegates are used, they too require wiring in IB
- You'd probably want to add some error checking to the code shown here if using it in a published application

Game Kit

Game Kit

- The Apple documentation states that "the Game Kit framework provides features designed for game developers who want to connect users of different iPhones together"
- However, the Game Kit framework can be useful for non-game related applications

Features

- Peer-to-peer connectivity — allows your app to create an ad-hoc Bluetooth network between multiple iPhone OS devices
- In-game voice — allows your app to handle voice communication between two iPhones or iPod touches

Classes & Delegates

- The Game Kit framework is a small (albeit powerful) framework consisting of just a few classes & delegates...
  - GKPeerPickerController
  - GKPeerPickerControllerDelegate
  - GKSession
  - GKSessionDelegate
  - GKVoiceChatService
  - GKVoiceChatClient

GKPeerPickerController & Delegate

- GKPeerPickerController — a UI for discovering and connecting to other iPhone OS devices
- GKPeerPickerControllerDelegate — delegate for handling GKPeerPickerController

GKSession & Delegate

- GKSession — provides capability to create and manage ad-hoc networks
- GKSessionDelegate — manages session events like connection requests and state changes

GKVoiceChatService & Client

- GKVoiceChatService — facilitates connecting of 2 iPhones into a voice chat
- GKVoiceChatClient — protocol which provides network connection, identifies peers and accepts/rejects voice chat requests

Device Support

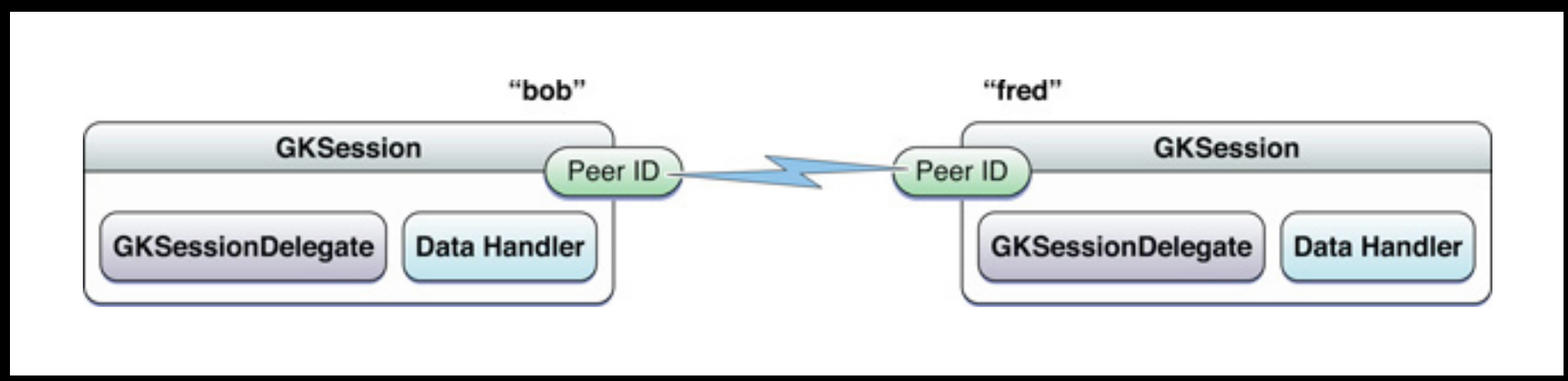
- Bluetooth networking is not supported on the original iPhone or the first-generation iPod touch
- Bluetooth networking is also not supported in the Simulator

Sessions

- When a session is created it connects to the network and discovers other sessions
- This session is used to transmit and receive data from other iPhone OS devices
- Your app implements a delegate to handle requests and receive data to hand off to your application

Peers

- iPhones connected to the ad-hoc network are known as peers
- Synonymous with a session running inside your application
- Each session creates a unique peerID used to identify the device on the network
- Communication between peers are done by using their peerID



Peer IDs and Names

- The peerID's are not intended to be exposed to end user's as they are just a long number like 1865470369
- This session method give you a user-friendly name for a peer...

```
- (NSString *)displayNameForPeer:(NSString *)peerID;
```

Peer Session States

- Peers may appear in a variety of states...

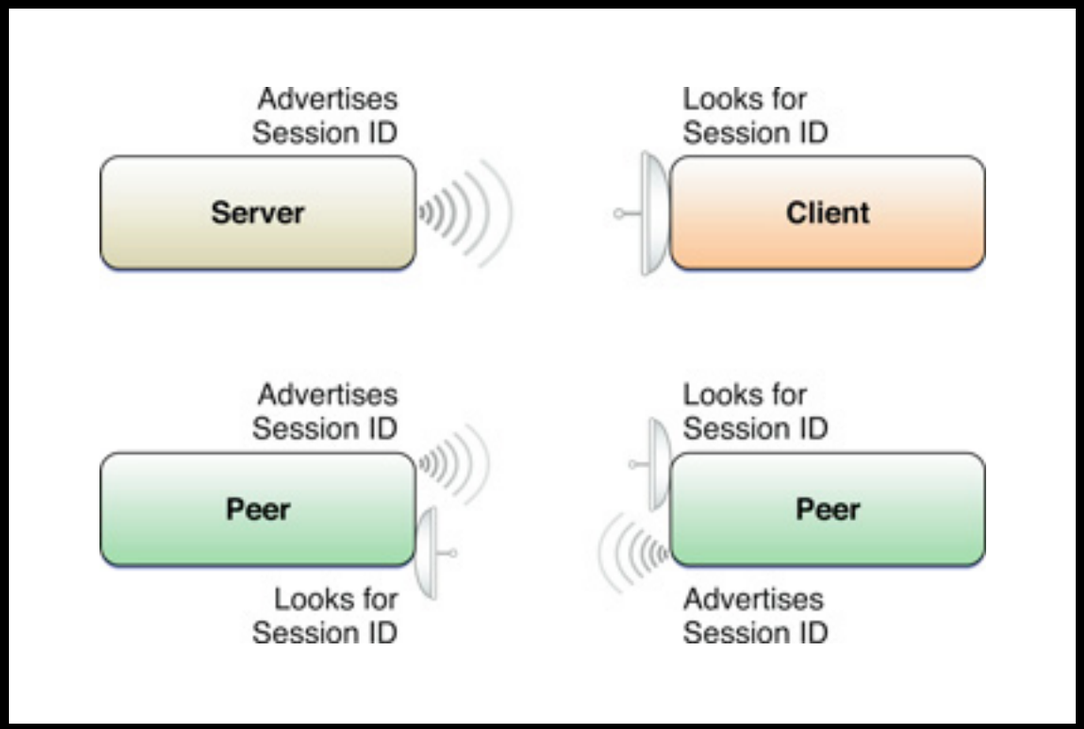
```
typedef enum {
  // not connected to session, but available
  // for connectToPeer:withTimeout:
  GKPeerStateAvailable,
  // no longer available
  GKPeerStateUnavailable,
  // connected to the session
  GKPeerStateConnected,
  // disconnected from the session
  GKPeerStateDisconnected,
  // waiting for accept, or deny response
  GKPeerStateConnecting,
} GKPeerConnectionState;
```

- These peer states are reported by the following delegate method...

```
- (void)session:(GKSession *)session
           peer:(NSString *)peerID
 didChangeState:(GKPeerConnectionState)state;
```

Discovering Peers

- Sessions discover other peers based on a session mode (set at initialization)
- Your app can configure the session to be...
  - A server — advertises a service type on the network
  - A client — searches for advertising servers
  - A peer, which advertises like both a server and a client



Implementing a Server

- An instance of your app acting as a server initializes a session by calling...

```
- (id)initWithSessionID:(NSString *)sessionID
                displayName:(NSString *)name
                sessionMode:(GKSessionMode)mode;
```

- Valid session modes are GKSessionModeServer or GKSessionModePeer
- Advertise the session by setting isAvailable property to YES

Implementing a Server

- Server is notified of client connection requests via a call to the delegate method...

```
- (void)session:(GKSession *)session
        didReceiveConnectionRequestFromPeer:(NSString *)peerID;
```

- Accept or reject the connection by calling...

```
- (BOOL)acceptConnectionFromPeer:(NSString *)peerID
                           error:(NSError **)error;
- (void)denyConnectionFromPeer:(NSString *)peerID;
```

- When successfully established, the following delegate method is called...

```
- (void)session:(GKSession *)session
           peer:(NSString *)peerID
 didChangeState:(GKPeerConnectionState)state;
```

Implementing a Client

- An instance of your app acting as a client initializes a session by calling...

```
- (id)initWithSessionID:(NSString *)sessionID
                displayName:(NSString *)name
                sessionMode:(GKSessionMode)mode;
```

- Valid session modes are GKSessionModeClient or GKSessionModePeer
- Your app then searches the network for advertised servers where isAvailable is YES

Implementing a Client

- When a client discovers an available server, it calls..

```
- (void)session:(GKSession *)session
           peer:(NSString *)peerID
 didChangeState:(GKPeerConnectionState)state;
```

- This method provides the peerID of the server, which you can use present a selection to the user
- Once the user selects a peer, the app then calls the following to actually open the connection...

```
- (void)connectToPeer:(NSString *)peerID
          withTimeout:(NSTimeInterval)timeout;
```

- When successfully established, it again calls -session:peer:didChangeState

Exchanging Data

- Data to be exchanged needs to be converted to an NSData
- Send data to all peers using...

```
- (BOOL)sendDataToAllPeers:(NSData *) data
                  withDataMode:(GKSendDataMode)mode
                         error:(NSError **)error;
```

- Or, just send data to select peers using...

```
- (BOOL)sendData:(NSData *) data
          toPeers:(NSArray *)peers
     withDataMode:(GKSendDataMode)mode
            error:(NSError **)error;
```

Exchanging Data

- For best performance, it is recommended that the size of the data objects be kept under 1k in length
- Larger messages (up to 95k) may need to be split and reassembled, incurring additional latency and overhead

Disconnecting Peers

- When you're ready to end a session, you should call...

```
- (void)disconnectFromAllPeers;
```

- You can disconnect a specific peer with the following...

```
- (void)disconnectPeerFromAllPeers:(NSString *)peerID;
```

- If a peer is unresponsive for a period of time its automatically disconnected
  - This duration can be set with the disconnectTimeout property
- You can detect peer disconnections using...

```
- (void)session:(GKSession *)session
           peer:(NSString *)peerID
 didChangeState:(GKPeerConnectionState)state;
```

Cleaning Up

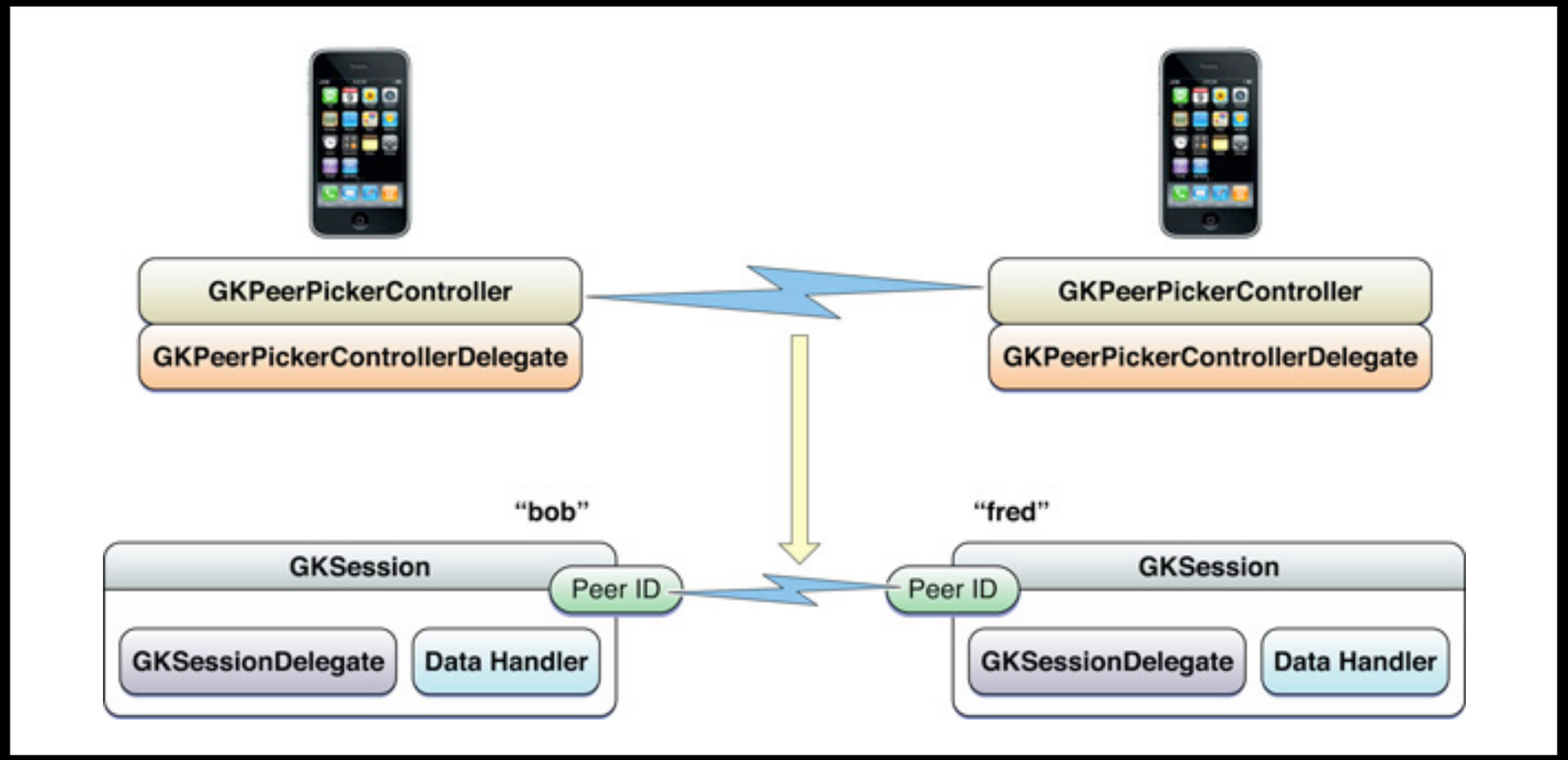
- When you are ready to end a session...
  - Disconnect from all peers
  - Set isAvailable to NO
  - Remove the data handler and delegate
  - Release the session

Peer Picker

- You may choose to implement your own UI using GKSession's delegate...
- Or, you may simply use Game Kit's provided UI for performing discovery and opening connections
- Game Kit's GKPeerPickerController class provides a common UI which may be used across apps

GKPeerPickerController

- Presents the user interface & responds to the user's actions
- Results in a fully configured GKSession that connects the two peers
- Customizable via GKPeerPickerControllerDelegate



GKPeerPickerController

- Needs to be configured with a connection mask using...

```
@property(nonatomic,assign) GKPeerPickerConnectionType connectionTypesMask;
```

- Can be one of.....

```
enum {
  // Online (Internet) based multiplayer connection
  GKPeerPickerConnectionTypeOnline = 1 << 0,
  // Nearby (Bluetooth) based multiplayer connection
  GKPeerPickerConnectionTypeNearby = 1 << 1
};
typedef NSUInteger GKPeerPickerConnectionType;
```

- The iPhone OS handles Bluetooth networking for you, however if you wish to do Internet networking, then you need to then perform whatever additional network setup, communication and tear-down that's required

GKPeerPickerController

- Once configured present the UI by calling the following controller method...

```
- (void)show;
```

- Once an item is selected, the following delegate method is called...

```
- (void)peerPickerController:(GKPeerPickerController *)picker
              didConnectPeer:(NSString *)peerID
                   toSession:(GKSession *)session;
```

- Your app then takes ownership of the session and should call the following to hide the dialog...

```
- (void)dismiss;
```

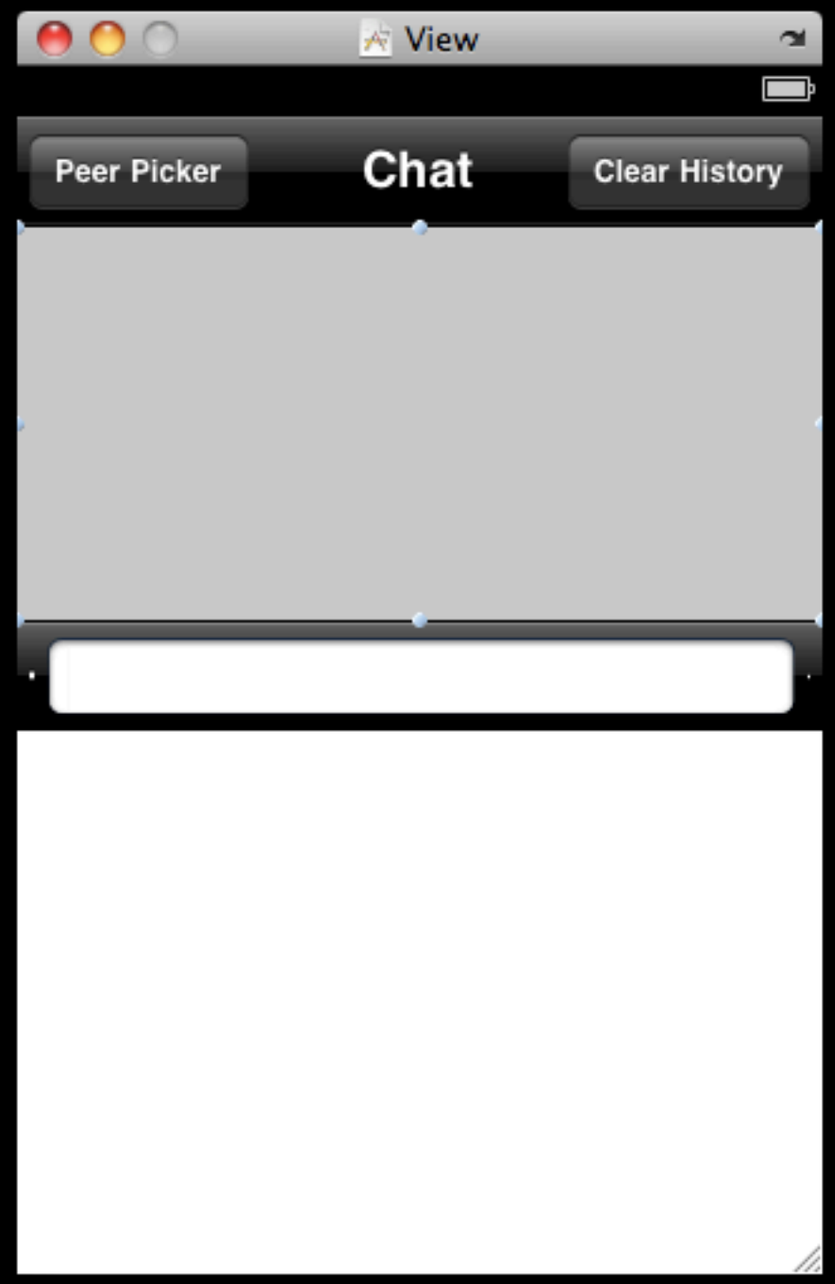
- If the user cancels, the following delegate method is called....

```
- (void)peerPickerControllerDidCancel:(GKPeerPickerController *)picker;
```

Demo

ChatViewController.xib

- Our UI consists of the following...
  - A toolbar with 2 buttons
    - One to bring up the peer picker
    - Another to clear the log
  - A large text view so we can log all of the chat messages
  - A text entry for sending messages
  - Some reserved space for the on screen keyboard



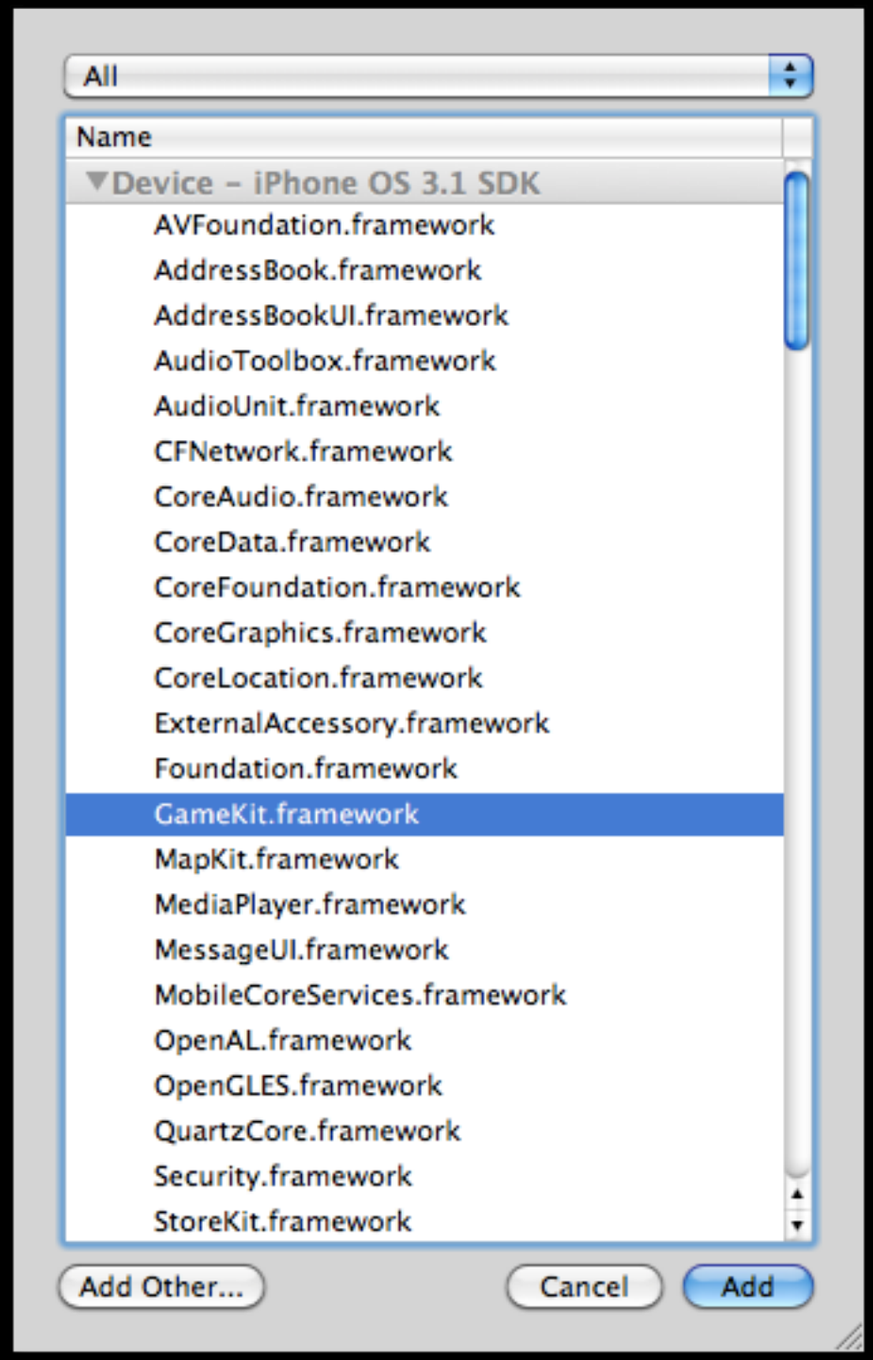
Changing the Status Bar Color

- The status bar style can be configured in the *-Info.plist file
- Simply add a field and select the “Status bar style” key and choose from one of the 3 options...

Key	Value
▼ Information Property List	(13 items)
Localization native development re	English
Status bar style ⇕	Opaque black style ⇕ +
Bundle display name	Gray style (default)
Executable file	Transparent black style (alpha of 0.5)
Icon file	Opaque black style
Bundle identifier	edu.umbc.${PRODUCT_NAME:rfc1034identifier}
InfoDictionary version	6.0
Bundle name	${PRODUCT_NAME}
Bundle OS Type code	APPL
Bundle creator OS Type code	????
Bundle version	1.0
Application requires iPhone enviror	☑
Main nib file base name	MainWindow

Adding the Game Kit Framework

- All of Game Kit's classes and delegates live within the Game Kit framework
- As such, it needs to be added to the project



ChatViewController.h

```
#import <GameKit/GameKit.h>
#import <UIKit/UIKit.h>

@interface ChatViewController : UIViewController <GKPeerPickerControllerDelegate,
                    GKSessionDelegate, UITextFieldDelegate, UITextViewDelegate> {

    UITextView *messagesView;
    UITextField *messageField;
    NSMutableArray *messages;
    GKPeerPickerController *picker;
    GKSession *session;
}

@property(nonatomic, retain) IBOutlet UITextView *messagesView;
@property(nonatomic, retain) IBOutlet UITextField *messageField;
@property(nonatomic, retain) NSMutableArray *messages;
@property(nonatomic, retain) GKPeerPickerController *picker;
@property(nonatomic, retain) GKSession *session;

- (IBAction)clear;
- (IBAction)disconnect;
- (IBAction)send;

@end
```

ChatViewController.m

```
#import "ChatViewController.h"

@implementation ChatViewController

@synthesize messagesView;
@synthesize messageField;
@synthesize messages;
@synthesize picker;
@synthesize session;
```

```
- (void)updateMessages {
  self.messagesView.text = [self.messages componentsJoinedByString:@"\n"];
  int contentHeight = self.messagesView.contentSize.height;
  int containerHeight = self.messagesView.bounds.size.height;
  if (contentHeight > containerHeight) {
    CGPoint bottom = CGPointMake(0, contentHeight - containerHeight);
    [self.messagesView setContentOffset:bottom animated:NO];
  }
}

- (IBAction)clear {
  [self.messages removeAllObjects];
  [self updateMessages];
}

// ...
```

ChatViewController.m

```
// ...

- (IBAction)connect {
  self.picker = [[[GKPeerPickerController alloc] init] autorelease];
  self.picker.delegate = self;
  [self.picker show];
}

- (IBAction)disconnect {
  [session disconnectFromAllPeers];
  session.available = NO;
  [session setDataReceiveHandler: nil withContext: nil];
  session.delegate = nil;
  self.session = nil;
  [self connect];
}

- (IBAction)send {
  NSData *data = [self.messageField.text dataUsingEncoding:NSUTF8StringEncoding];
  [session sendDataToAllPeers:data withDataMode:GKSendDataReliable error:nil];
  NSString *formatted = [NSString stringWithFormat:@"%@: %@", [self.session
                  displayNameForPeer:self.session.peerID], self.messageField.text];
  [self.messages addObject:formatted];
  [self updateMessages];
}

// ...
```

ChatViewController.m

```
// ...

- (void)viewWillAppear:(BOOL)animated {
  [super viewWillAppear:animated];
  [self.messageField becomeFirstResponder];
  [self connect];
}

- (void)viewDidLoad {
  self.messages = [NSMutableArray array];
}

- (void)peerPickerController:(GKPeerPickerController *)p
              didConnectPeer:(NSString *)peerID
                   toSession:(GKSession *) sess {
  self.session = sess;
  self.session.delegate = self;
  [self.session setDataReceiveHandler: self withContext:nil];
  [self.picker dismiss];
}

// ...
```

ChatViewController.m

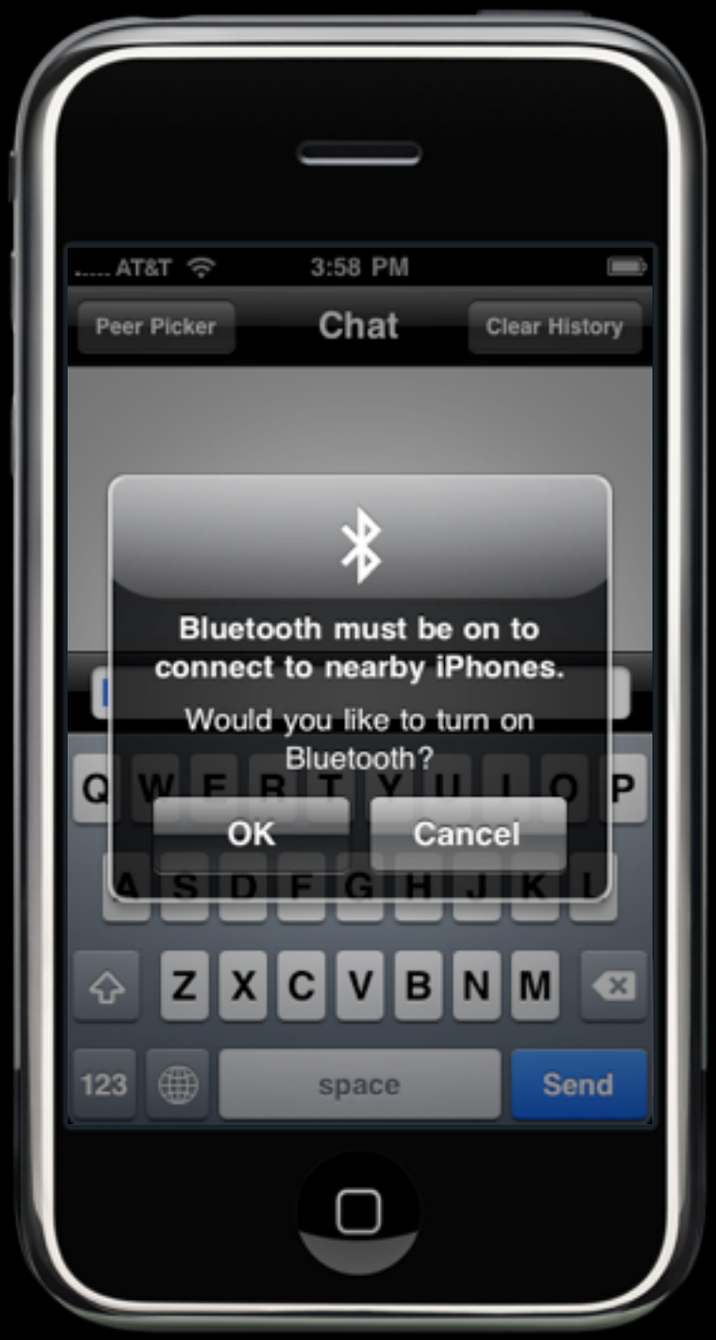
```
// ...

- (void)session:(GKSession *)session
           peer:(NSString *)peerID
 didChangeState:(GKPeerConnectionState)state {
  if (state == GKPeerStateDisconnected) {
    [self disconnect];
  }
}

- (BOOL)textFieldShouldReturn:(UITextField *)textField {
  [self send];
  self.messageField.text = @"";
  return YES;
}

- (BOOL)textView:(UITextView *)textView
shouldChangeTextInRange:(NSRange)range
 replacementText:(NSString *)text {
  return NO;
}

// ...
```

ChatViewController.m

```
// ...

- (void) receiveData:(NSData *)data
            fromPeer:(NSString *)peer
           inSession:(GKSession *)session
             context:(void *)context {
  NSString *message = [[[NSString alloc] initWithData:data
                                             encoding:NSUTF8StringEncoding] autorelease];
  NSString *formatted = [NSString stringWithFormat:@"%@: %@",
                                 [self.session displayNameForPeer:peer], message];
  [self.messages addObject:formatted];
  [self updateMessages];
}

- (void)dealloc {
  self.messagesView = nil;
  self.messageField = nil;
  self.picker = nil;
  self.session = nil;
  self.messages = nil;
  [super dealloc];
}

@end
```

The Resulting App

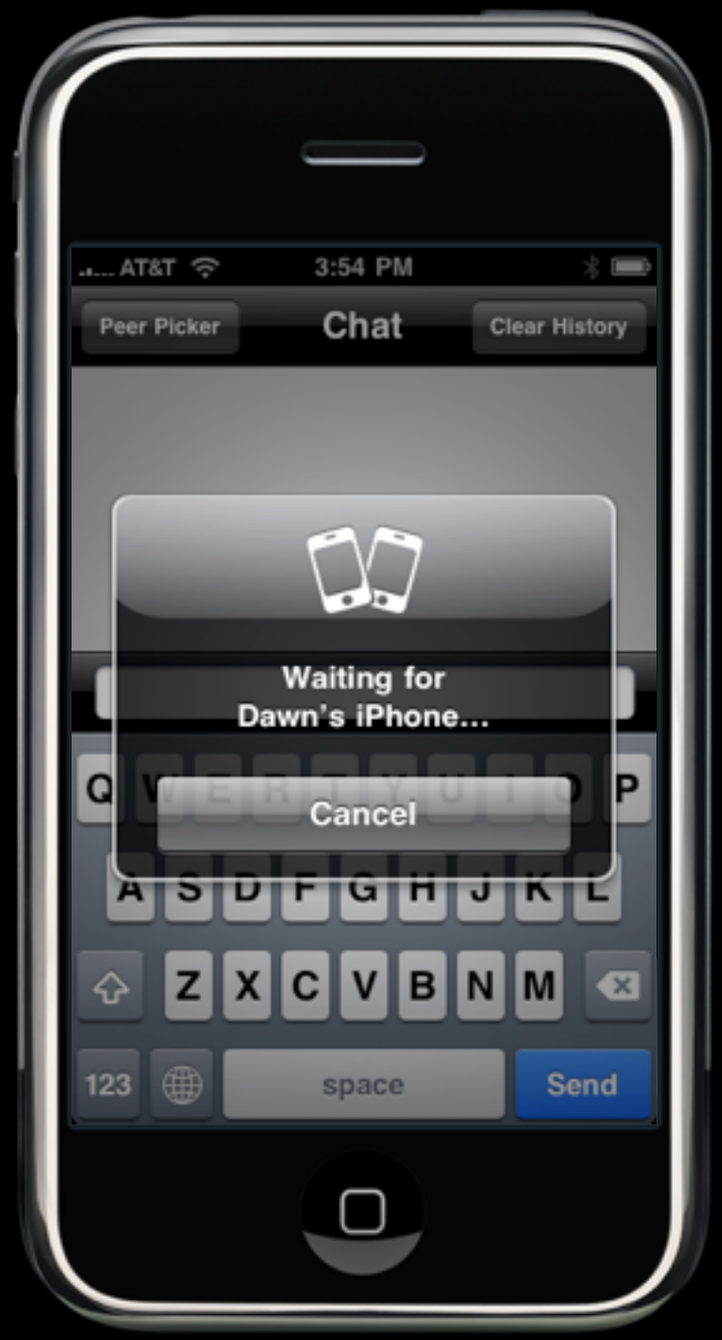
- If using the Peer Picker and Bluetooth services are turned off, the API is smart enough to ask the user if they with to turn it on

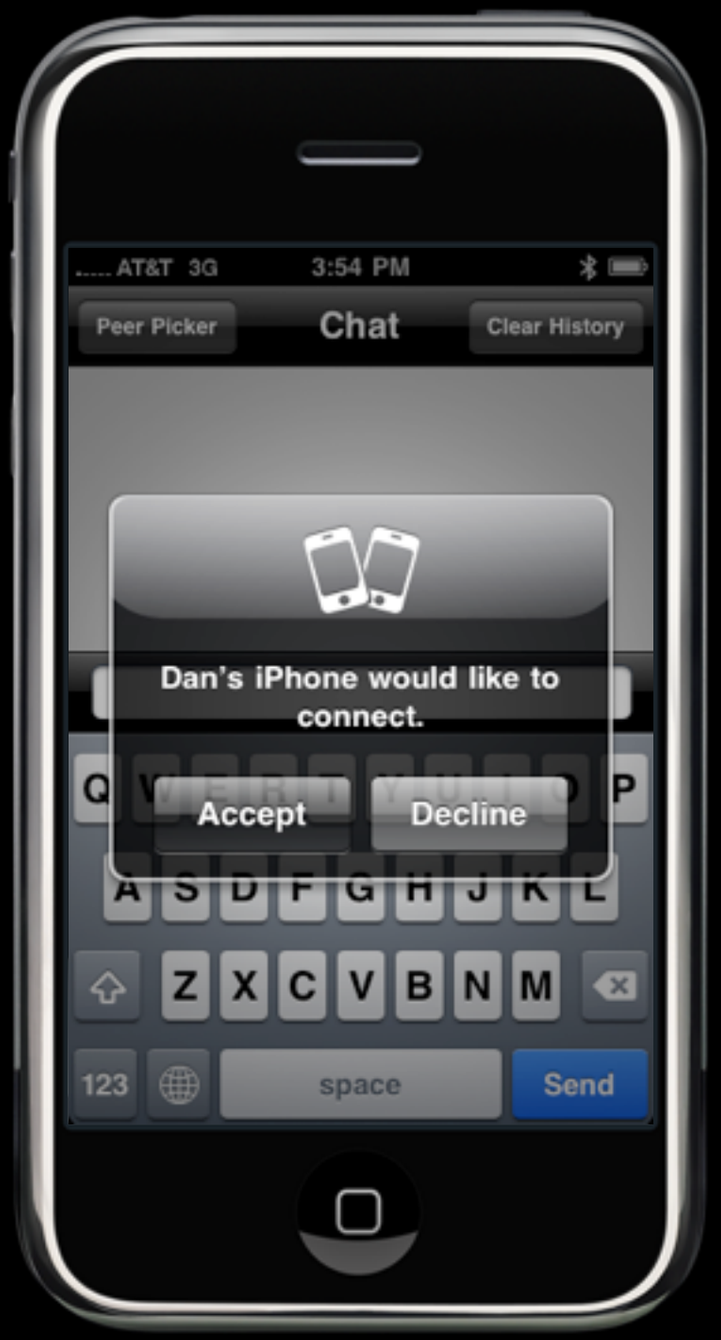


The Resulting App

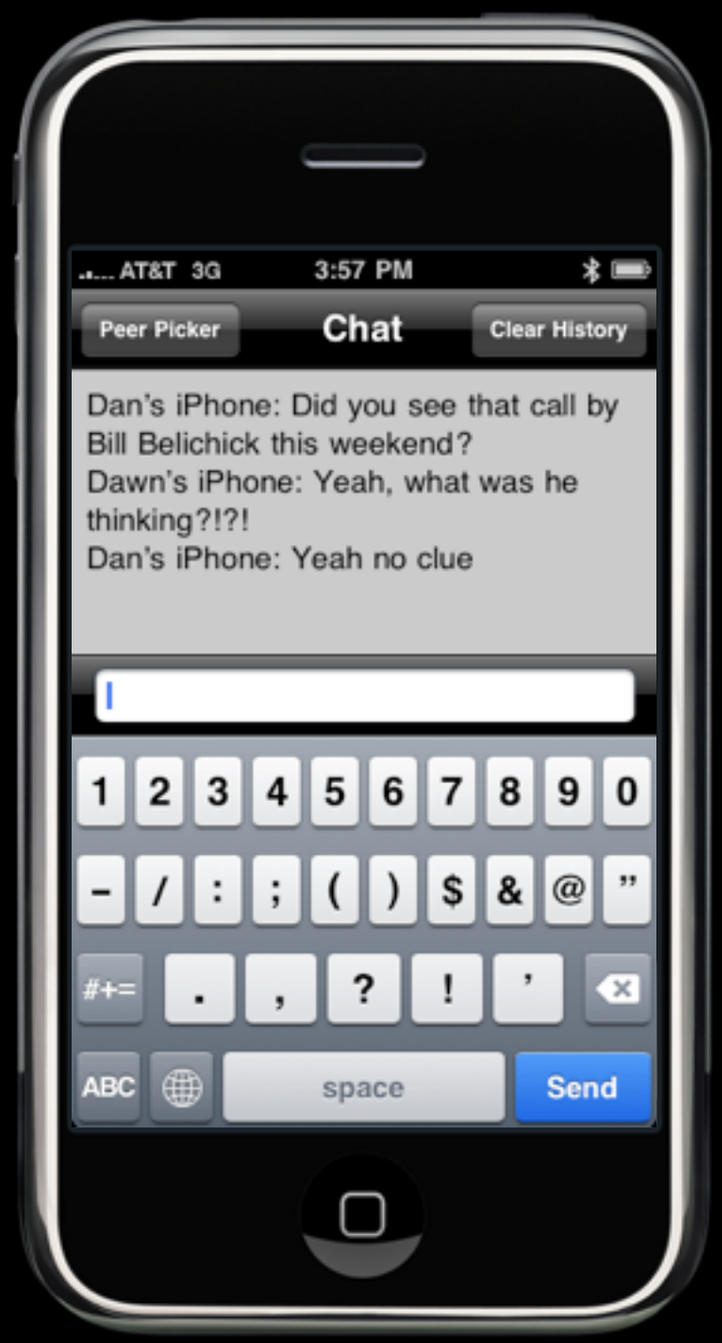
The Resulting App

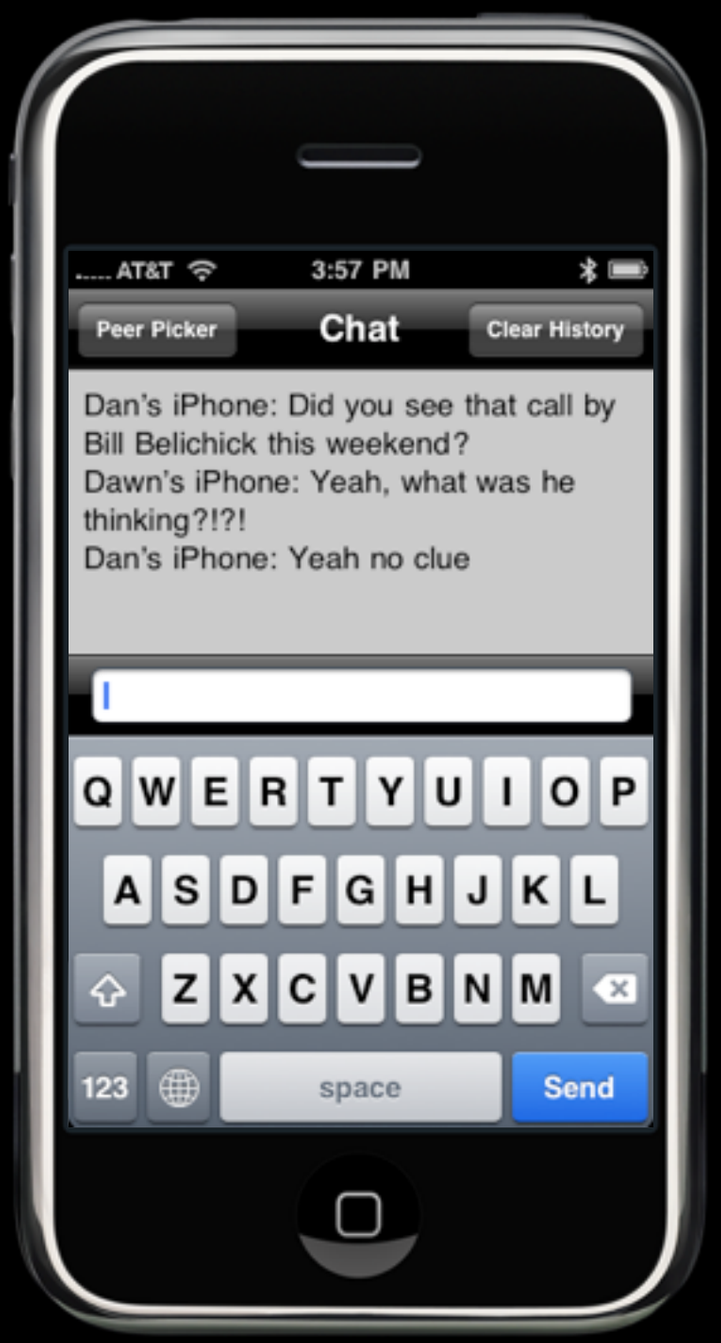
The Resulting App





The Resulting App

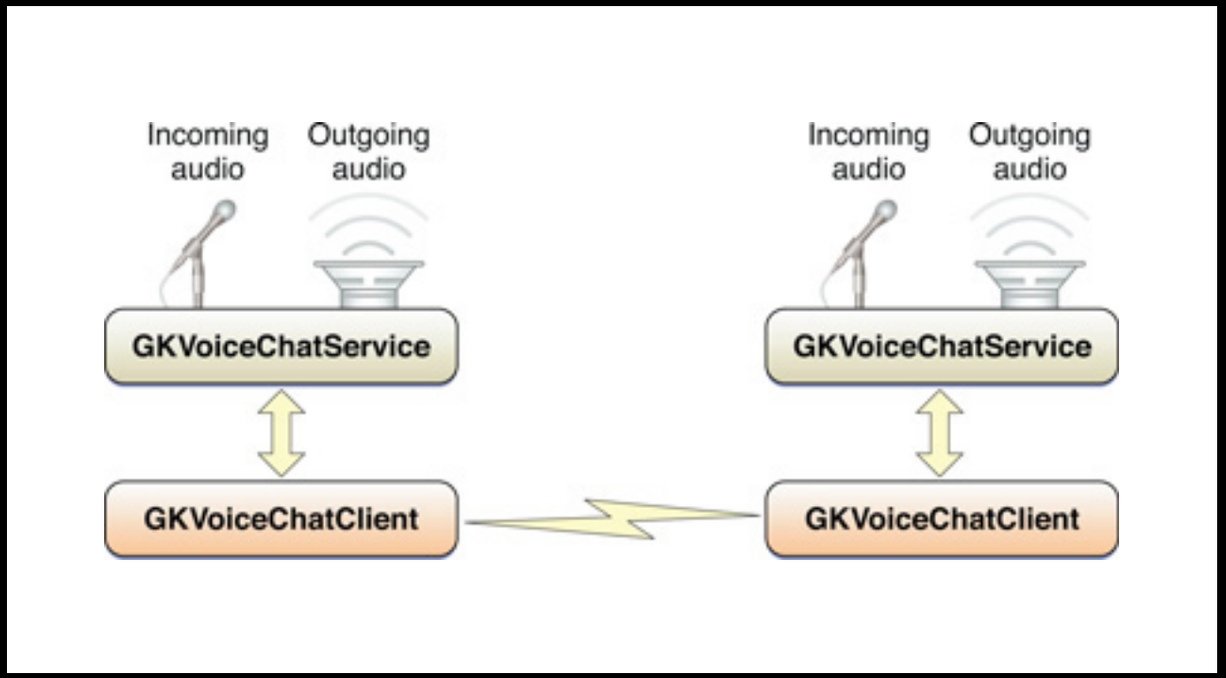




In-Game Voice

GKVoiceChatService

- Allows your app to create a voice chat between two iPhone OS devices
- The voice chat service samples the microphone and plays audio received from the other peer



GKVoiceChatClient

- In-game voice requires your app to provide a client (can be the view controller like other delegates) to implement the GKVoiceChatClient protocol
- The primary responsibility of the client is to connect the two participants together so that the voice chat service can exchange configuration data

Participant Identifiers

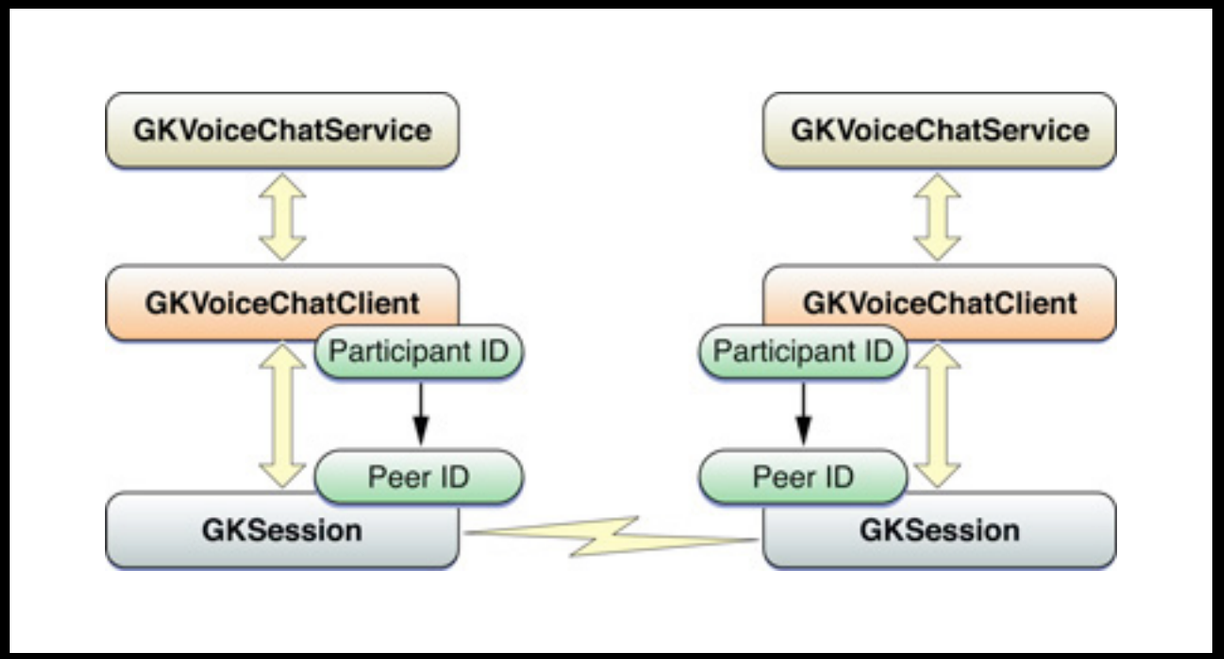
- Each participant in a voice chat is identified by a unique participant identifier string provided by your client
- The format and meaning of a participant identifier string is left to your client to decide

Discovering Other Participants

- The chat service uses the client's network connection to exchange config data between the peers in order to create a direct connection between the two
- The voice chat service does not provide a mechanism to discover the participant identifier of other participants
  - Typically piggy-back on an already established connection
- Your app is responsible for providing the participant IDs of other peers and translating these identifiers into connections to other participants

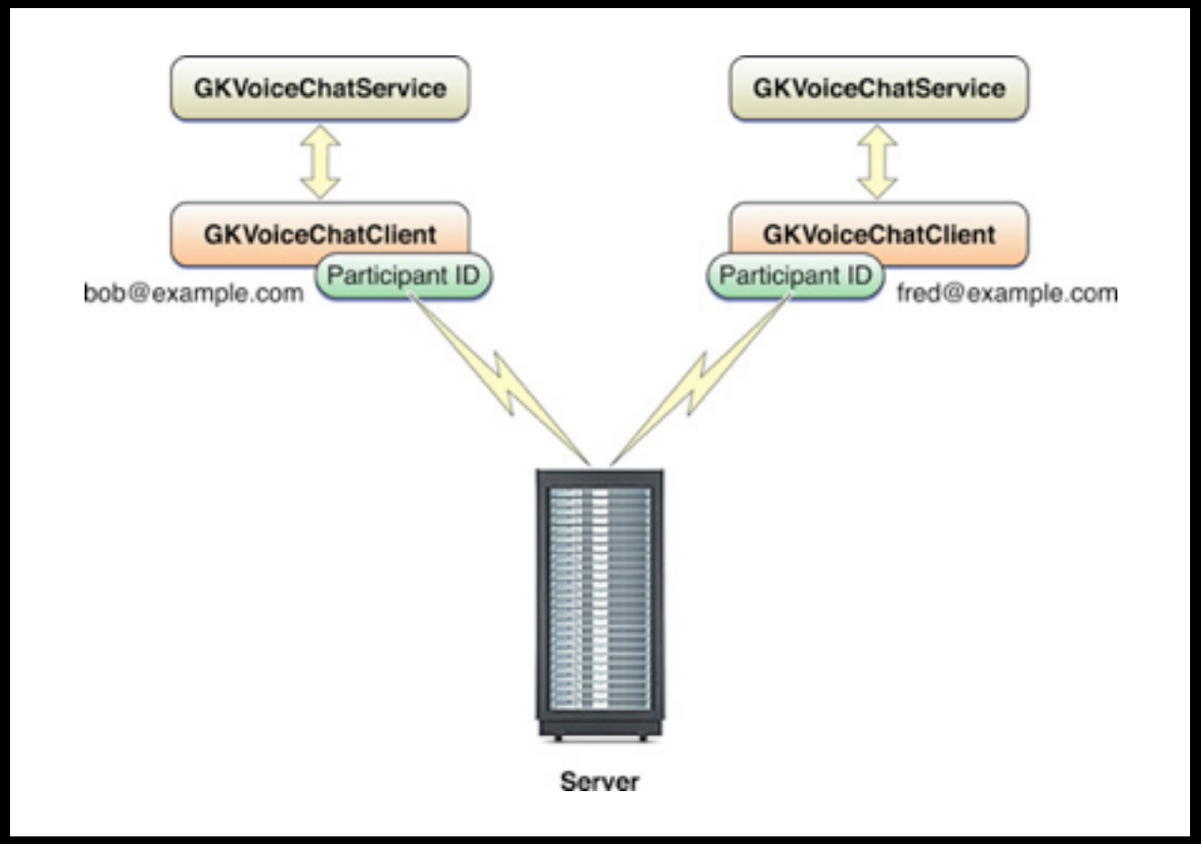
Using an Existing Session

- If your application is already connected to another device through a GKSession object, then each peer on the network is already uniquely identified by a peerID
- The client could simply reuse each peer's ID as the participant identifier and use the session to send and receive data...



Server Based Discovery

- If the 2 devices are not directly connected, your app needs another service to facilitate the cross-discovery of the two participants to provide connectivity
- For example, you might use a directory server to lookup other devices and provide connectivity



Starting a Chat

- To start a voice chat, one of the participants calls the following voice chat service's method specifying the ID of the other participant

```
- (BOOL)startVoiceChatWithParticipantID:(NSString *)participantID
                                  error:(NSError **)error;
```

- When a service receives a connection request, the following client method is called to handle it...

```
- (void)voiceChatService:(GKVoiceChatService *)voiceChatService
didReceiveInvitationFromParticipantID:(NSString *)participantID
                              callID:(NSInteger)callID;
```

Starting a Chat

- The client can accept the chat request by calling the following service method...

```
- (BOOL)acceptCallID:(NSInteger)callID
                error:(NSError **)error;
```

- Or the client can reject the connection by calling...

```
- (void)denyCallID:(NSInteger)callID;
```

- Once a connection is established, the following client method is invoked...

```
- (void)voiceChatService:(GKVoiceChatService *)voiceChatService
didStartWithParticipantID:(NSString *)participantID;
```

Stopping a Chat

- To stop a chat, your app should call the following service method...

```
- (void)stopVoiceChatWithParticipantID:(NSString *)participantID;
```

- Your app should also stop the chat if it discovers that the other user is no longer available

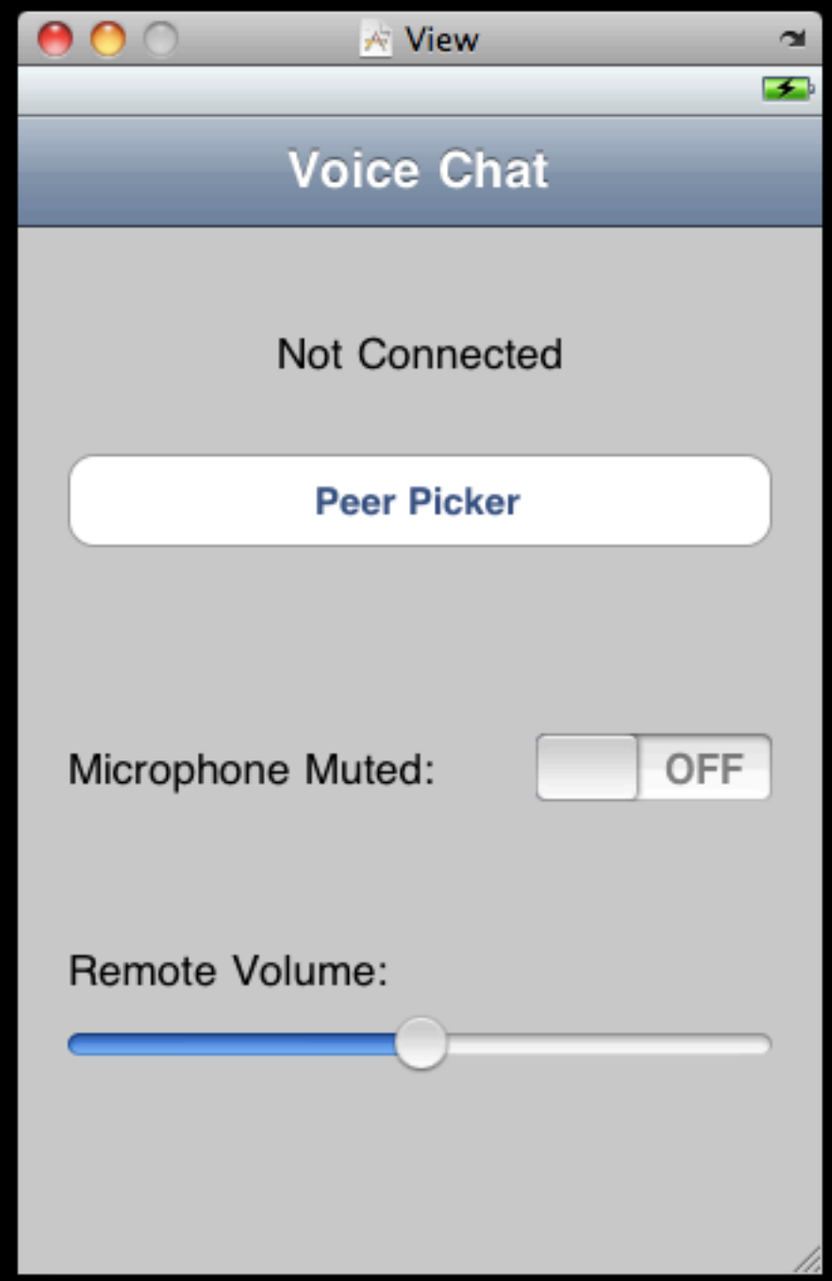
Controlling the Chat

- Once the participants are connected, speech is automatically transmitted between the two iPhones OS devices
- Your application can control certain characteristics of the chat including...
  - Muting the device's microphone by setting the service's microphoneMuted property
  - Adjust the volume of the remote participant by setting the service's remoteParticipantVolume property
  - Ability to get monitoring data for both local and remote voice streams

Demo

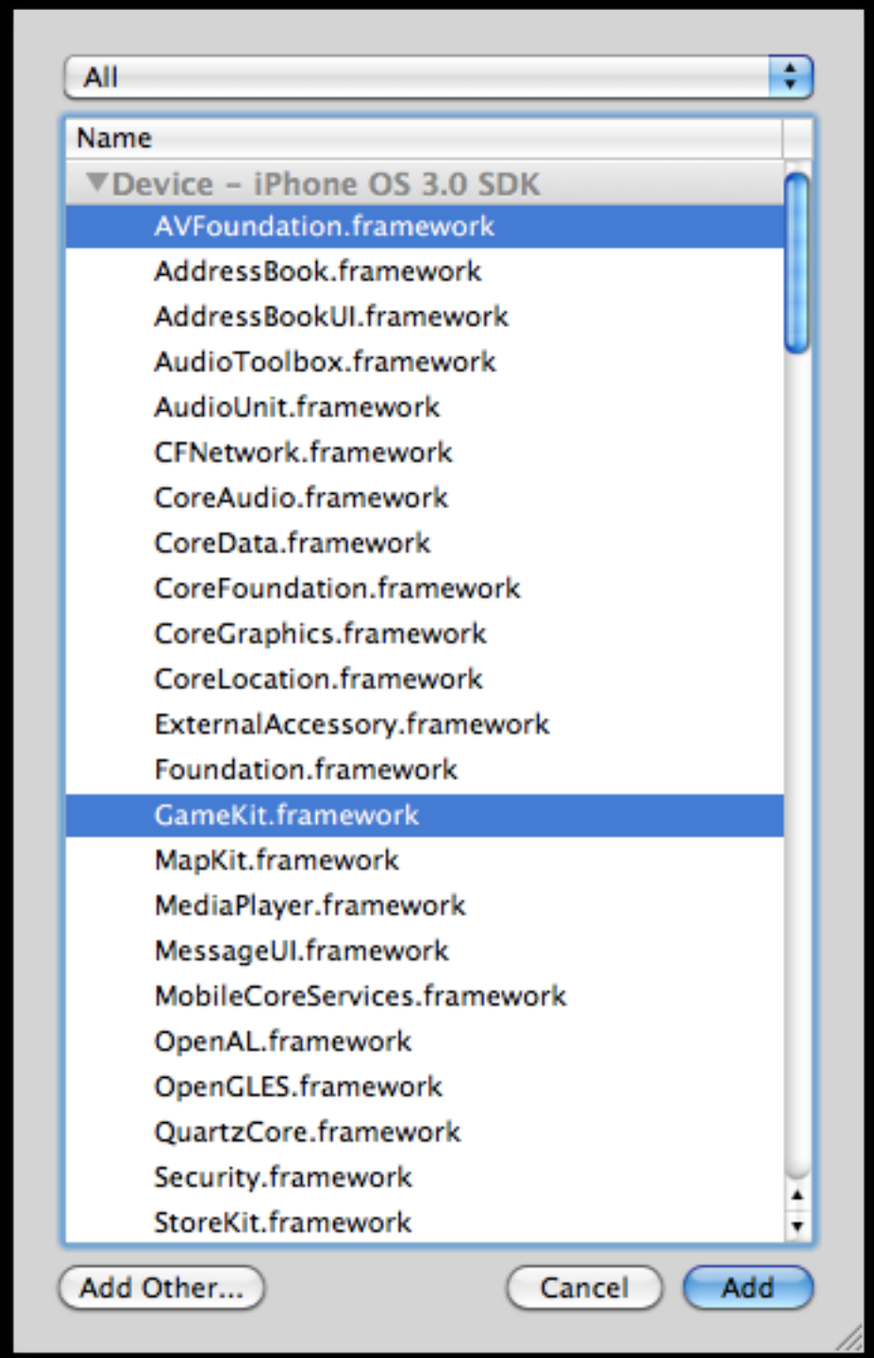
VoiceChatViewController.xib

- Our UI consists of the following...
  - A status label to tell what peer (if any) we're connected to
  - A button to bring up the peer picker
  - A switch to toggle the microphone's mute on and off
  - A volume slider to adjust the remote volume level



Adding the Necessary Frameworks

- For this app, we need to include the following frameworks
  - Game Kit
  - AVFoundation



VoiceChatViewController.h

```
#import <GameKit/GameKit.h>
#import <UIKit/UIKit.h>

@interface VoiceChatViewController : UIViewController
            <GKPeerPickerControllerDelegate, GKSessionDelegate, GKVoiceChatClient> {

  GKPeerPickerController *picker;
  GKSession *session;
  UISwitch *microphoneToggle;
  UILabel *status;

}

@property(nonatomic, retain) GKPeerPickerController *picker;
@property(nonatomic, retain) GKSession *session;
@property(nonatomic, retain) IBOutlet UISwitch *microphoneToggle;
@property(nonatomic, retain) IBOutlet UILabel *status;

- (IBAction)disconnect;
- (IBAction)toggleMute;
- (IBAction)setVolume:(id)sender;

@end
```

VoiceChatViewController.m

```
#import "VoiceChatViewController.h"

@implementation VoiceChatViewController

@synthesize picker;
@synthesize session;
@synthesize muteMicrophone;
@synthesize status;

- (void)connect {
  self.picker = [[[GKPeerPickerController alloc] init] autorelease];
  self.picker.delegate = self;
  [self.picker show];
}

// ...
```

VoiceChatViewController.m

```
// ...

- (IBAction)disconnect {
  self.status.text = @"Not Connected";
  [session disconnectFromAllPeers];
  session.available = NO;
  [session setDataReceiveHandler: nil withContext: nil];
  session.delegate = nil;
  self.session = nil;
  if (self.muteMicrophone.on) {
    [self toggleMute];
  }
  [self connect];
}

- (IBAction)toggleMute {
  [[GKVoiceChatService defaultVoiceChatService]
    setMicrophoneMuted:self.muteMicrophone.on];
}

// ...
```

VoiceChatViewController.m

```
// ...

- (IBAction)setVolume:(id)sender {
  UISlider *slider = sender;
  [[GKVoiceChatService defaultVoiceChatService]
   setRemoteParticipantVolume:slider.value];
}

- (void)peerPickerController:(GKPeerPickerController *)p
              didConnectPeer:(NSString *)peerID
                   toSession:(GKSession *)sess {
  self.session = sess;
  self.session.delegate = self;
  [self.session setDataReceiveHandler: self withContext:nil];
  [self.picker dismiss];
}

// ...
```

VoiceChatViewController.m

```
// ...

- (void)session:(GKSession *)session
           peer:(NSString *)peerID
 didChangeState:(GKPeerConnectionState)state {
  switch (state) {
    case GKPeerStateConnected:
      [GKVoiceChatService defaultVoiceChatService].client = self;
      [[GKVoiceChatService defaultVoiceChatService]
       startVoiceChatWithParticipantID:peerID error: nil];
      self.status.text = [NSString stringWithFormat:@"Connected to %@",
                          [self.session displayNameForPeer:peerID]];
      break;
    case GKPeerStateDisconnected:
      [[GKVoiceChatService defaultVoiceChatService]
       stopVoiceChatWithParticipantID:peerID];
      [self disconnect];
      break;
    default:
      NSLog(@"%s - state = %d", __FUNCTION__, state);
  }
}

// ...
```

VoiceChatViewController.m

```
// ...

- (NSString *)participantID {
  return session.peerID;
}

- (void)voiceChatService:(GKVoiceChatService *)voiceChatService
                sendData:(NSData *)data
         toParticipantID:(NSString *)participantID {
  [self.session sendData:data toPeers:[NSArray arrayWithObject: participantID]
   withDataMode:GKSendDataReliable error: nil];
}

- (void) receiveData:(NSData *)data
            fromPeer:(NSString *)peer inSession:(GKSession *)session
             context:(void *)context {
  [[GKVoiceChatService defaultVoiceChatService] receivedData:data
   fromParticipantID:peer];
}

// ...
```

VoiceChatViewController.m

```
// ...

- (void)viewDidLoad {
  [super viewDidLoad];
  [[GKVoiceChatService defaultVoiceChatService] setRemoteParticipantVolume:.5];
  [self connect];
}

- (void)dealloc {
  self.picker = nil;
  self.muteMicrophone = nil;
  self.session = nil;
  self.status = nil;
  [super dealloc];
}

@end
```

Audio Sessions

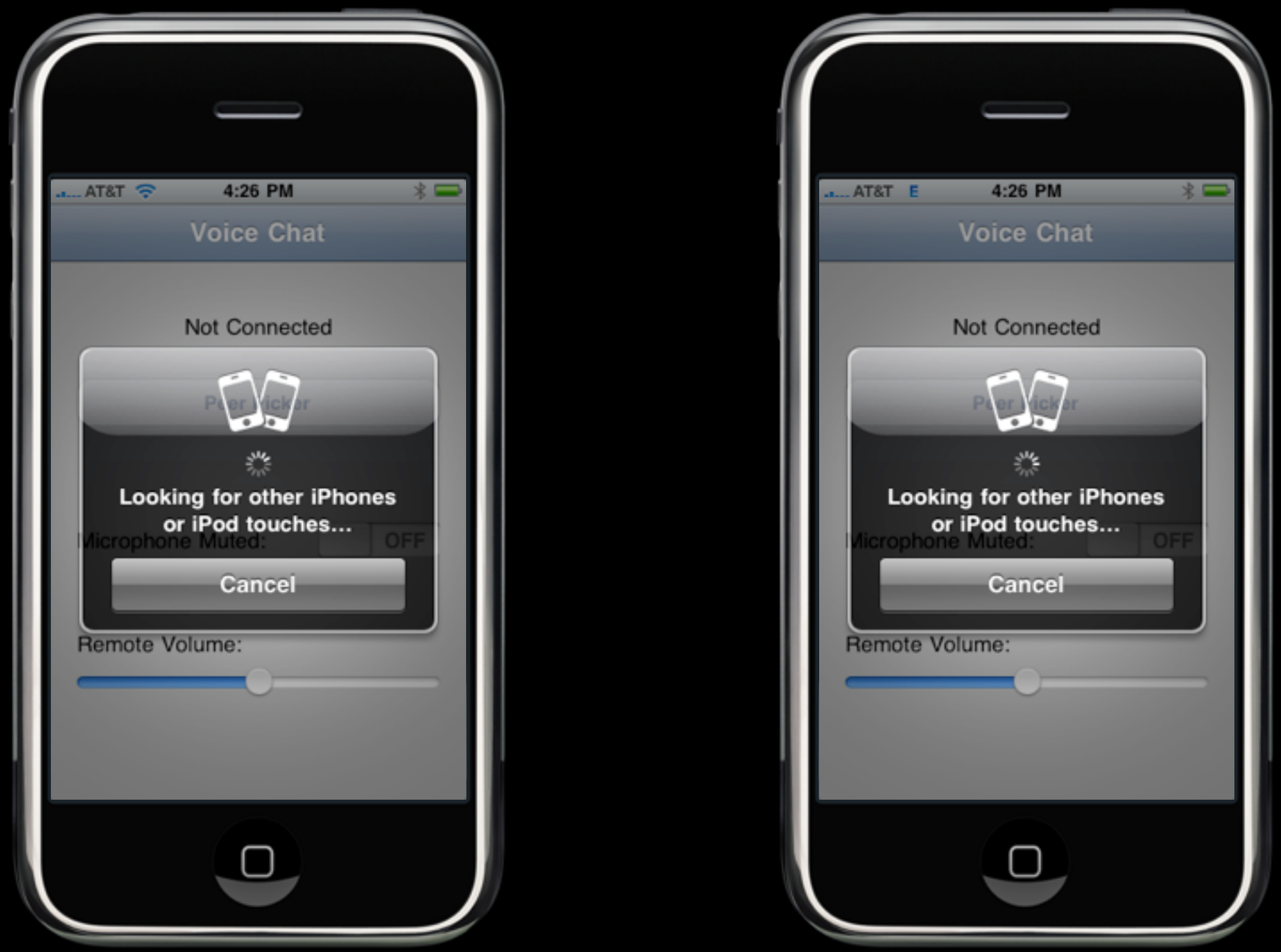
- We want to make sure we disable currently playing audio (such as the iPod app) when our app starts
- We can use the AVAudioSession which has different categories that define the audio behavior of our app
  - For example, if we declare our app as being capable of playing and recording, we can disable the iPod app when we start up
- The app delegate has good lifecycle points to hook into

VoiceChatAppDelegate.m

```
#import <AVFoundation/AVFoundation.h>

#import "VoiceChatAppDelegate.h"
#import "VoiceChatViewController.h"

@implementation VoiceChatAppDelegate

@synthesize window;
@synthesize viewController;
```

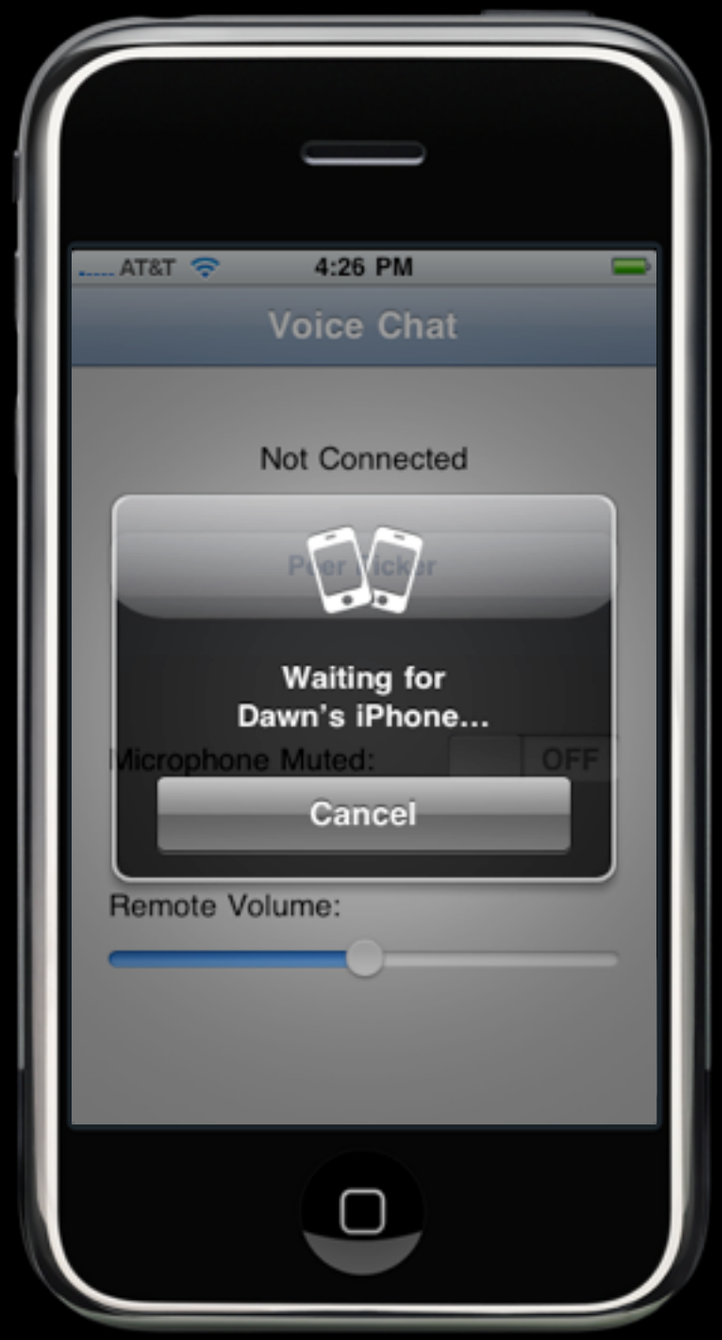
```
- (void)applicationDidFinishLaunching:(UIApplication *)application {
  [window addSubview:viewController.view];
  [window makeKeyAndVisible];
  AVAudioSession *audioSession = [AVAudioSession sharedInstance];
  [audioSession setCategory:AVAudioSessionCategoryPlayAndRecord error:nil];
  [audioSession setActive:YES error:nil];
}

- (void)dealloc {
  [viewController release];
  [window release];
  [super dealloc];
}

@end
```

The Resulting App

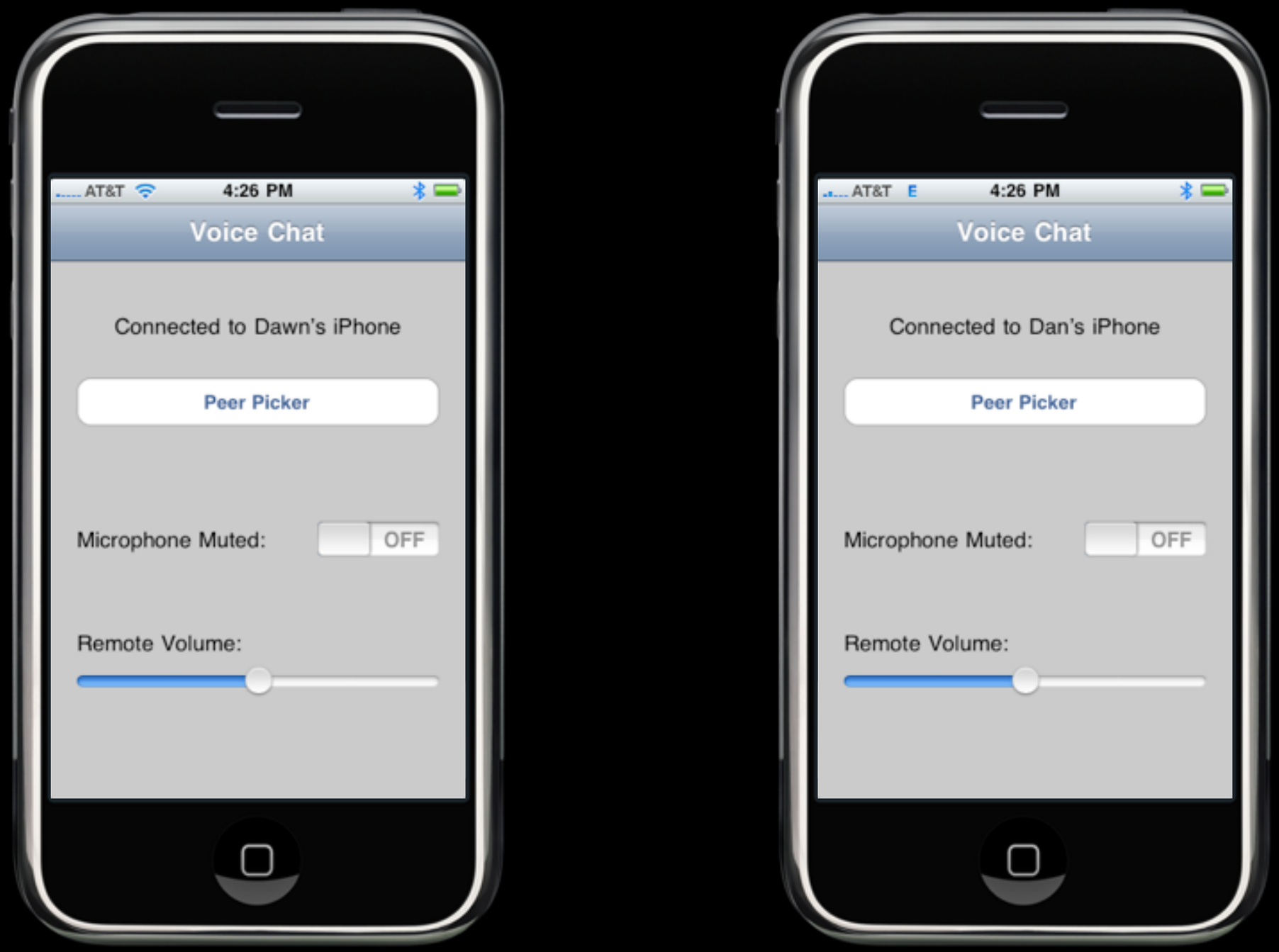


The Resulting App

The Resulting App



The Resulting App



Additional Resources

- Game Kit Programming Guide
  - http://developer.apple.com/IPhone/library/documentation/NetworkingInternet/Conceptual/GameKit_Guide/

For Next Class

- "iPhone Application Programming Guide" sections...
  - Playing Media Items with iPod Library Access
  - Using Video in iPhone OS
  - Taking Pictures with the Camera
  - Picking a Photo from the Photo Library
- "Address Book Programming Guide for iPhone OS"
  - http://developer.apple.com/iphone/library/documentation/ContactData/Conceptual/AddressBookProgrammingGuideforiPhone/