CS 47

Beginning iPhone Application Development

Week 8: Notifications, Audio/Video

Office Hours

- Saturday, March 13th
- 10am-11am
- Redrock Cafe, 2nd floor

Agenda

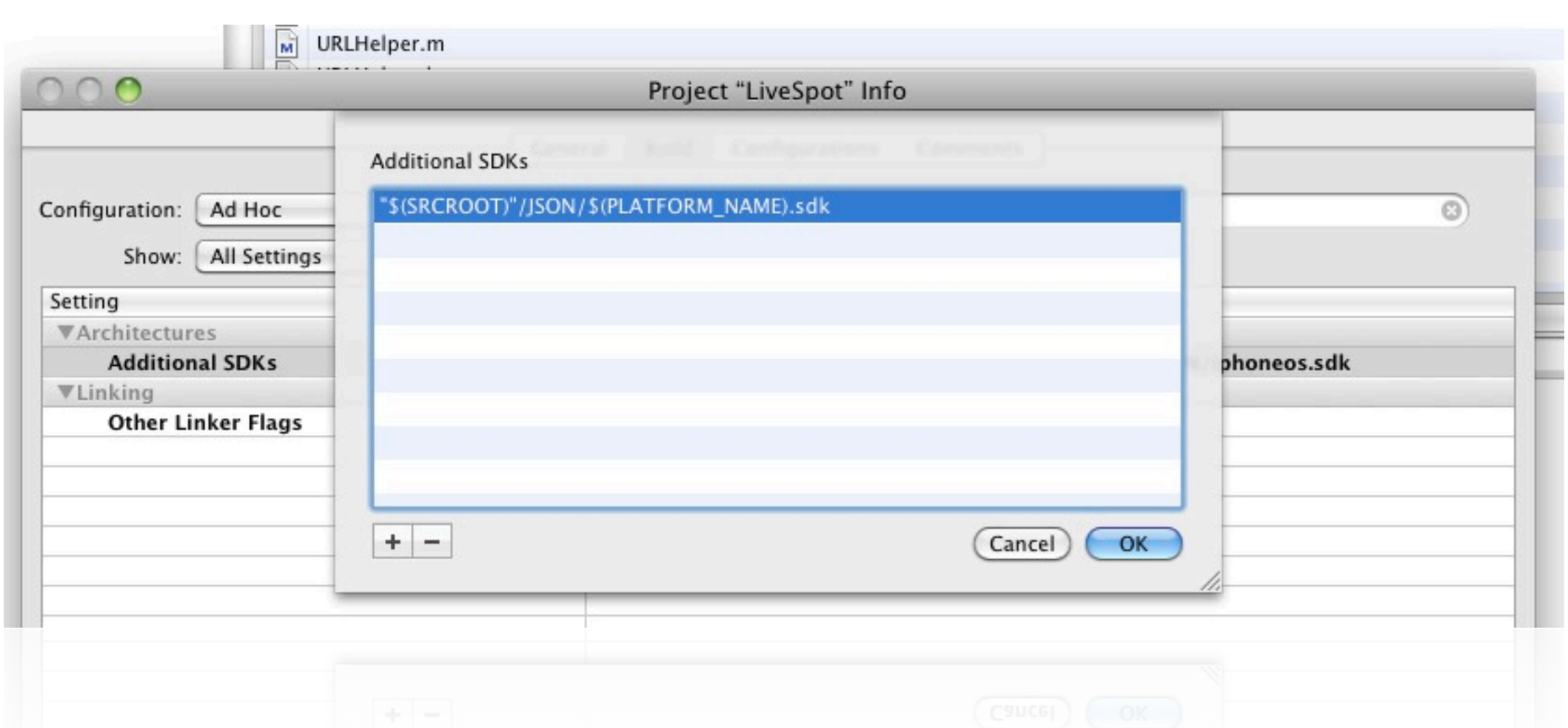
- Some Tips
- AVFoundation
- AudioToolkit
- MPMoviePlayerController
- MPMusicPlayerController

Importing Libraries

- Xcode targets the i386 architecture for the simulator, and ARM for the device
- Your project can make use of third-party library/framework/SDKs - they should provide support for both architectures
- Make sure you import the right architecture for your target

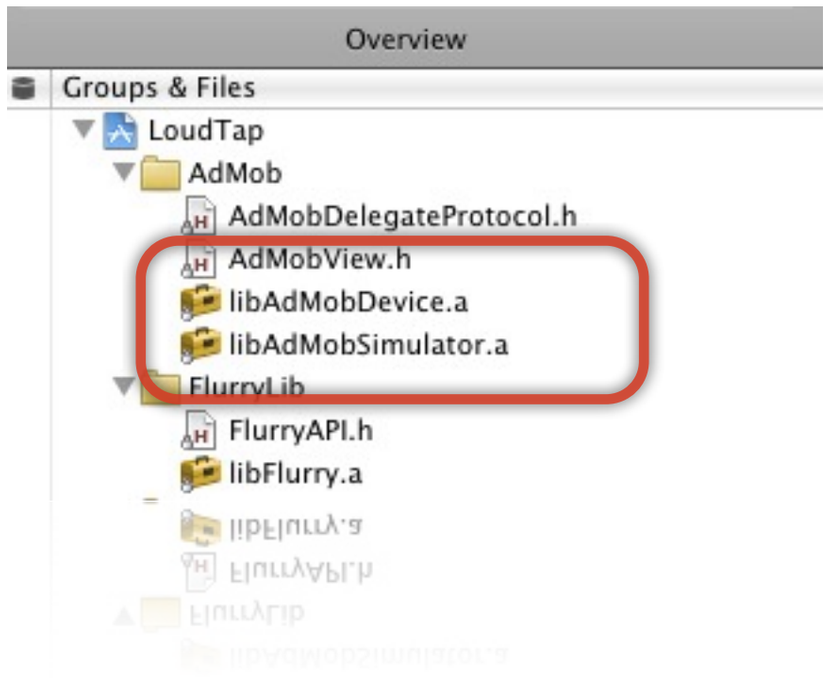
Importing Libraries

- Use the $PLATFORM_NAME variable in include paths (e.g. in the Library Search Paths field or the additional SDKs field)



Importing Libraries

- Or just include both at once
- You'll get a warning about the "wrong" architecture, but you can ignore it



The ‘id’ Return

- Many functions have an ‘id’ return type, e.g. [NSDictionary objectForKey:..]
- This means that it can return any Objective-C class! The compiler will not do type checking

```
// An NSNumber value is returned and put in an NSString
// NO COMPILER WARNING
NSString *myString = [dic objectForKey:@”Some_NSNumber”];
```

- You will get a runtime exception if you try to call an NSString selector on the object

The ‘id’ Return

- In general it’s ok to assume the returned object will be a certain class, especially if it’s a documented function
- Just be careful. Use the `isKindOfClass` selector or class_getName to interogate an object if you have to

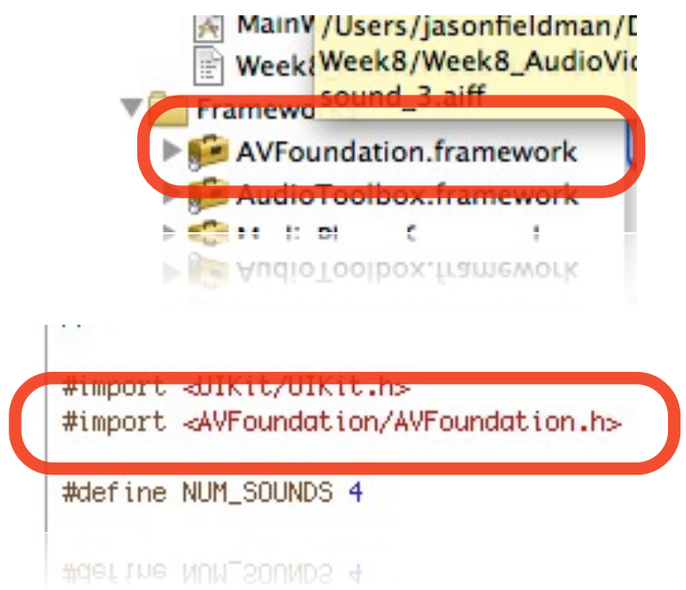
Use Backtrace!

- Whenever you get an EXC_BAD_ACCESS, runtime exception or any other application crash, be sure to analyze the backtrace
  - Open the debug console (Run->Console, or Command-Shift-R - enter “bt”

```
Program received signal:  "EXC_BAD_ACCESS".
(gdb) bt
#0  0x9217aed4 in objc_msgSend ()
#1  0x00002b8a in -[SoundboardViewController init] (self=0x451eb10, _cmd=0x92c3e4f8) at /Users/jasonfieldman/Documents/CS47/Week8/Week8_AudioVideo/Classes/
    SoundboardViewController.m:59
#2  0x0000241a in -[Week8_AudioVideoAppDelegate applicationDidFinishLaunching:] (self=0x45053e0, _cmd=0x9338df3a, application=0x4119000) at /Users/
    jasonfieldman/Documents/CS47/Week8/Week8_AudioVideo/Classes/Week8_AudioVideoAppDelegate.m:36
#3  0x002a25cb in -[UIApplication _performInitializationWithURL:sourceBundleID:] ()
#4  0x002ab9b6 in -[UIApplication _runWithURL:sourceBundleID:] ()
#5  0x002a8b34 in -[UIApplication handleEvent:withNewEvent:] ()
#6  0x002a467f in -[UIApplication sendEvent:] ()
#7  0x002ab061 in _UIApplicationHandleEvent ()
#8  0x02624d59 in PurpleEventCallback ()
#9  0x01f41b80 in CFRunLoopRunSpecific ()
#10 0x01f40c48 in CFRunLoopRunInMode ()
#11 0x002a2e15 in -[UIApplication _run] ()
#12 0x002abfaf in UIApplicationMain ()
#13 0x00002240 in main (argc=1, argv=0xbfffef90) at /Users/jasonfieldman/Documents/CS47/Week8/Week8_AudioVideo/main.m:14
(gdb)
```

AVFoundation

- Ok, let's start with audio!
- There are many different ways to get an iPhone app to generate and record audio, but AVFoundation is the best compromise between simplicity and functionality
- AVAudioPlayer
- AVAudioRecorder



AVAudioPlayer

- AVAudioPlayer handles the playback of any type of supported audio file (wav, mp3, etc) with minimal configuration
- Very easy setup:

```
NSURL *fileURL = ...; /* Point to audio file */

AVAudioPlayer *player =
  [[AVAudioPlayer initWithContentsOfURL:fileURL error:nil];
  /* Ready to play! */
```

- (The file has to be on the device!)

AVAudioPlayer

- You can also load the AVAudioPlayer from memory

```
NSData *data = ...; /* binary data of an audio file */

AVAudioPlayer *player =
  [[AVAudioPlayer initWithData:data error:nil];
  /* Ready to play! */
```

- Good for playing sound clips that are loaded from a remote source, or playing dynamically-created files

AVAudioPlayer

- Each AVAudioPlayer instance can only play a single audio file!
- That audio file is determined at object-creation time. You cannot make an AVAudioPlayer play new audio data after it has been created
- You need to create an AVAudioPlayer object for each unique sound

AVAudioPlayer

- Also holds true for playing multiple simultaneous instances of the same sound - each needs its own AVAudioPlayer
- You only need one AVAudioPlayer per sound if you are ok stopping playback of existing instances (so only one instance of each sound is playing at a time)

AVAudioPlayer

- Simple programmatic playback control

```
[player play];
[player stop];
[player pause];
[player prepareToPlay];       /* Preloads data */
player.numberOfLoops = ...;   /* Set # of loops */

if (player.isPlaying) ...     /* Check if we're active */

player.duration               /* How long is the audio? */
player.currentTime            /* Current playing offset (RW) */
```

AVAudioPlayer

- If you anticipate playing the same sound multiple times (halting existing playback), do this to the existing AVAudioPlayer object:

```
[player stop];
player.currentTime = 0; /* Dynamic time shifting */
[player play];
```

AVAudioPlayer

- Each AVAudioPlayer object has individual control over its own volume

```
player.volume = 0.5; /* 0.0 to 1.0, r/w */
```

- You can use this to equalize different audio clips
- Volume can be set dynamically while the audio is playing

AVAudioPlayer

- Get basic audio level data

```
player.meteringEnabled = YES;
...
while loop {
  [player updateMeters];
  float average = [player averagePowerForChannel:0];
  float peak    = [player    peakPowerForChannel:0];
}
```

- Returns dB: -160dB [silent] to 0dB [loud]
- Only turn it on if you need it... needs extra processing power

AVAudioPlayer

- Use an AVAudioPlayerDelegate to catch events `(player.delegate = self;)`

```
- audioPlayerDidFinishPlaying:successfully:
- audioPlayerDecodeErrorDidOccur:error:
- audioPlayerBeginInterruption:
- audioPlayerEndInterruption:
```

AVAudioPlayer

- The AVAudioPlayer object needs to be "alive" during the entire sound playback
- It's setup for easy management of persistent objects (i.e. if you have an array of AVAudioPlayers that will sit around for awhile)
- But what if you just want one-off objects? (why? memory constraints, simultaneous)

AVAudioPlayer

- Use the delegate to free the object

```
{ ...
  player = [[AVAudioPlayer alloc] initWithURL:... error:nil];
  player.delegate = self;
  [player play];
  /* Needs to be released when finished */
...}


- (void) audioPlayerDidFinishPlaying:(AVAudioPlayer*)player
                        successfully:(BOOL)success {
  [player autorelease];
}
```

AVAudioPlayer

- What if you want multiple AVAudioPlayer objects to play the same sound more than once simultaneously, but you also want persistent objects?
- Create an array of AVAudioPlayer objects just for that one sound (make sure they all point to the same NSData object)

AVAudioPlayer

```
@interface MyClass : NSObject {
  AVAudioPlayer soundArray[kMaxSimultaneous];
  int           currentIndex;
}

@implementation MyClass

- (id) init { ...
  NSData *d = ...; /* Sound data */
  for (int i = 0; i < kMaxSimultaneous; i++) {
    soundArray[i] = [[AVAudioPlayer alloc] initWithData:d error:nil];
  }
...}

- (void) playSound {
  [soundArray[currentIndex] stop];
  soundArray[currentIndex].currentTime = 0;
  [soundArray[currentIndex] play];
  currentIndex = (currentIndex + 1) % kMaxSimultaneous;
}

@end
```

AVAudioPlayer

- So what can't you do with the AVAudioPlayer?
    - Streaming - you need the entire file
    - Synchronization - you can't start audio at specific times
- If you need these, you'll have to use AudioToolkit (cry)

AVAudioRecorder

- Very similar API to AVAudioPlayer, but used for recording data onto disk
- Interestingly, there is no way to record into an NSMutableData object (why not?) - you have to record onto disk

```
recorder = [[AVAudioRecorder alloc] initWithURL:...
                                       settings:...
                                          error:nil];
```

AVAudioRecorder

- The URL must point to a **writable** file
- Make a path to your document or cache directories

```
/* Create recording path */
NSArray *paths = NSSearchPathForDirectoriesInDomains
                 (NSCachesDirectory, NSUserDomainMask, YES);
NSString *cacheDirectory = [paths objectAtIndex:0];
NSString *filePath =
        [NSString stringWithFormat:@"%@/recording.wav",
                                   cacheDirectory];
fileURL = [[NSURL alloc] initFileURLWithPath:filePath];
```

AVAudioRecorder

- Replaces the ‘play’ selector with ‘record’
- You cannot time shift the recorder (`currentTime` is a readonly parameter)
- Similar delegate and level-metering API

AVAudioRecorder

- Part of the initialization function is a settings dictionary

```
/* Record settings for typical WAV file */

NSMutableDictionary *settings = [NSMutableDictionary dictionary];

[settings setObject:[NSNumber numberWithInt:kAudioFormatLinearPCM]
             forKey:AVFormatIDKey];

[settings setObject:[NSNumber numberWithFloat:44100.0]
             forKey:AVSampleRateKey];
```

AVAudioRecorder

- When recording is finished (e.g. you call the `stop` method), the recorded data will be at the file URL you specified
- You can use that same URL to play back the recorded audio, compress it, or send it up to a remote server

AudioToolkit

- The AudioToolkit was the original "high level" audio API in the iPhone SDK
- It was such a nightmare to use that Apple released AVFoundation for iPhone OS 2.2 to make normal audio tasks easier
- But you still need AudioToolkit to do streaming and synchronization

AudioToolkit

Problem: AudioToolkit will destroy you.

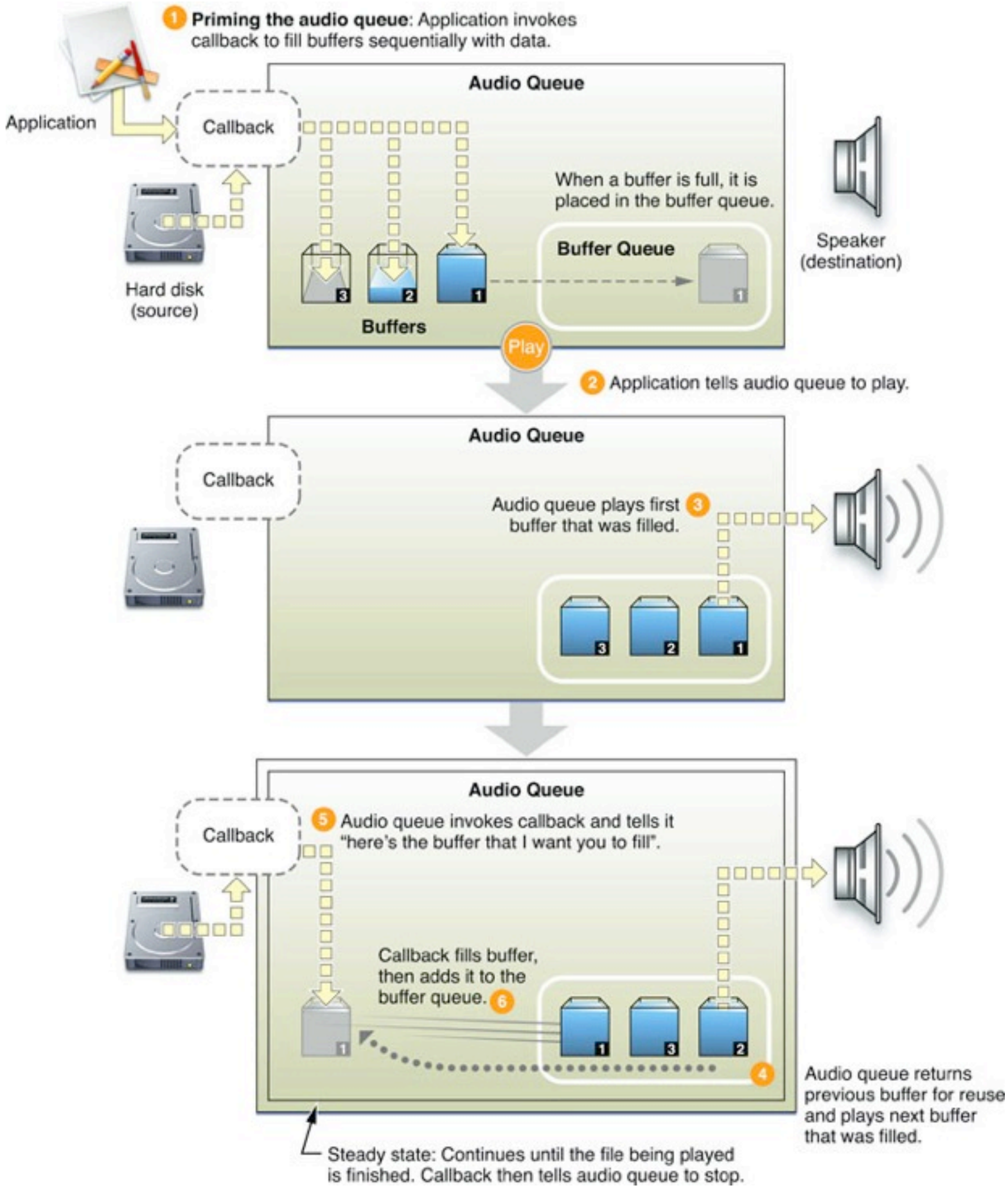
(We'll do a brief overview.)

AudioToolkit

- It's all about the AudioQueueRef object
- The AudioQueueRef is associated with several AudioQueueBuffer objects that make a circular data queue of audio packets
- The AudioQueueRef constantly reads data from the buffers and sends it to the hardware

AudioToolkit

- When the AudioQueueRef has exhausted data in an AudioQueueBuffer node, it triggers a callback that needs to refill the AudioQueueBuffer with new data


1 Priming the audio queue: Application invokes callback to fill buffers sequentially with data.
Application
Callback
Audio Queue
When a buffer is full, it is placed in the buffer queue.
Buffer Queue
Speaker (destination)
Hard disk (source)
3
2
1
Buffers
Play
2 Application tells audio queue to play.
Audio Queue
Callback
Audio queue plays first buffer that was filled. 3
Audio Queue
Callback
5 Audio queue invokes callback and tells it "here's the buffer that I want you to fill".
Callback fills buffer, then adds it to the buffer queue. 6
4
Audio queue returns previous buffer for reuse and plays next buffer that was filled.
Steady state: Continues until the file being played is finished. Callback then tells audio queue to stop.

AudioToolkit

- Where does the audio data come from?
  - AudioFileID (disk or memory)
  - AudioFileStreamID (streaming)
- Each of the above APIs has a method to get more data, and understands how to extract audio data packets from the binary file

AudioToolkit

- AudioFileID has a traditional pull API (AudioFileReadPackets) where you simply read more data from an existing source
- AudioFileStreamID has a "push" API, where you feed in data as it is streamed to you, and the AudioFileStreamID triggers a callback when completed packets have arrived

AudioToolkit

- So that's a really high level description of the AudioToolkit API.
- Read the documentation for AudioQueueRef, AudioFileID and AudioFileStreamID
- Read the Audio Queue Services Programming Guide for detailed examples
- Streaming example in this week's demo

Notifications

- The notification system provides synchronous message dispatching for various events
- It's a lot like the "addTarget" behavior of UI elements, except that it works for arbitrary events, and for any type of object
- Used by some of the multimedia APIs

Notifications

- Notifications are handled by the NSNotificationCenter object
- You'll want to get the default global instance:

```
NSNotificationCenter *center = [NSNotificationCenter defaultCenter];
```

Notifications

- To receive notifications, you need to register your instance as an observer

```
/* Called inside your controller class */

NSNotificationCenter *center = [NSNotificationCenter defaultCenter];

[center addObserver:self
           selector:@selector(handleNotification:)
               name:kNotificationIWantToObserve
             object:someObjectThatGeneratesNotifications];
```

- name or object can be nil (wildcards)

Notifications

- You should always remove yourself as an observer during deallocation, or when you are no longer interested in notifications

```
[center removeObserver:self];  /* Removes all observations */

[center removeObserver:self
                  name:kNotificationIWantToObserve
                object:someObjectThatGeneratesNotifications];
```

- name or object can be nil (wildcards)

Notifications

- If you are a notification generator, you can post arbitrary notifications:

```
- postNotification:
- postNotificationName:object:
- postNotificationName:object:userInfo:
```

Notifications

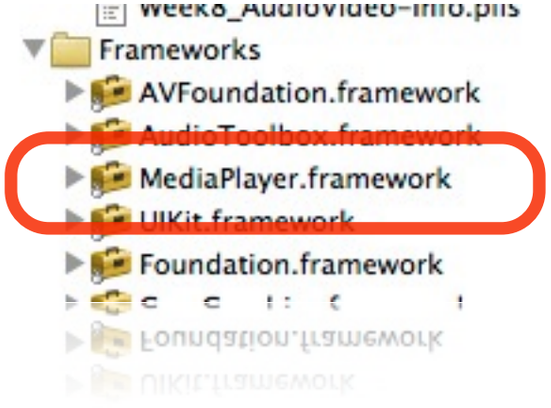
- The Notification Center will send notifications to all objects that match listening parameters
- Messages are posted synchronously! postNotification: does not return until all receivers complete their handlers
- Use NSNotificationQueues to send asynchronous notifications

Notifications

- You can see how this creates a very loose "cloud" to send event notifications around your application - good when you may want more than one receiver of an event
- You should still favor the protocol/delegate mechanism for strongly-bonded object relationships (much more explicit)

MPMoviePlayerController

- So you want to play a movie in your app?
- Use MPMoviePlayerController
- You can play local or remotely-streamed movies



MPMoviePlayerController

- Controller initialization is fairly typical:

```
MPMoviePlayerController *controller =
  [[MPMoviePlayerController alloc] initWithContentURL:...];
```

- Setup various properties before you play:

```
controller.backgroundColor  = ...;
controller.scalingMode      = ...;
controller.movieControlMode = ...;
```

MPMoviePlayerController

- Use the `play` and `stop` selectors to programmatically control playback
- Unlike other view controllers, you **do not** add this one to your view hierarchy!
  - i.e. **do not** present this as a modal view controller
- Calling the play command will automatically bring up a full-screen player

MPMoviePlayerController

- So how do you know when the movie is finished?

```
/* Assign finish notification handler */
[[NSNotificationCenter defaultCenter]
     addObserver:self
        selector:@selector(movieFinishedCallback:)
            name:MPMoviePlayerPlaybackDidFinishNotification
          object:moviePlayerController];
```

MPMoviePlayerController

- Use the notification to release the player

```
-(void) movieFinishedCallback:(NSNotification*)aNotification {
  /* Grab the movie controller object */
  MPMoviePlayerController* controller = [aNotification object];

  /* Remove notifications */
  [[NSNotificationCenter defaultCenter]
       removeObserver:self
                 name:MPMoviePlayerPlaybackDidFinishNotification
               object:controller];

  /* Release movie controller */
  [controller release];
}
```

MPMusicPlayerController

- So what about playing music from the device's media library?
- Use MPMusicPlayerController

MPMusicPlayerController

- You can access the iPod music controller:

```
MPMusicPlayerController *musicController =
            [MPMusicPlayerController iPodMusicPlayer];
```

- Any changes you make to the media playback state of this controller will affect the iPod controller on the device

MPMusicPlayerController

- Or you can setup a local media player that is disconnected from the state of the device iPod player

```
MPMusicPlayerController *musicController =
            [MPMusicPlayerController applicationMusicPlayer];
```

- Allows you to access the media library without changing user's generic iPod state

MPMusicPlayerController

- Unlike the AVAudioPlayer, the MPMusicPlayerController is designed to handle playlists (or, what they call Queues)

- Each entry in the playlist is represented by an MPMediaItem object

- So aside from standard `play`, `pause`, `stop`, it also provides `skipToNextItem` and `skipToPreviousItem`

MPMusicPlayerController

- Control playback mode and state

```
player.repeatMode
player.shuffleMode
player.volume

/* Direct time shifting */
player.currentPlaybackTime

/* Set this while stopped/paused */
player.nowPlayingItem

/* Read-only */
player.playbackState
```

MPMusicPlayerController

- You can get the currently playing item with `nowPlayingItem` - gives you back an MPMediaItem object (or nil)
- Use `valueForProperty` to get attribute info, e.g.

```
[item valueForProperty:MPMediaItemPropertyTitle];
[item valueForProperty:MPMediaItemPropertyAlbum];
[item valueForProperty:MPMediaItemPropertyPlaybackDuration];
```

MPMusicPlayerController

- Here is something truly bone-headed: You can only interrogate the currently playing item
- You cannot programmatically interrogate other elements of the current queue (playlist)
- So there is no way to display playlist info of a MPMusicPlayerController just through its own API

MPMusicPlayerController

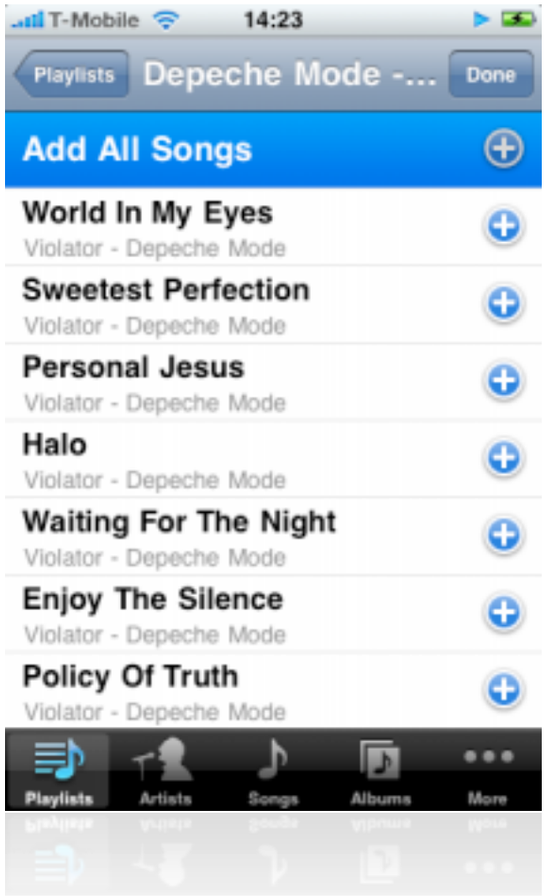
- You can only display playlist information if you gather it from the MPMediaPickerController

```
MPMediaPickerController *picker = [[[MPMediaPickerController alloc]
                  initWithMediaTypes:MPMediaTypeAny] autorelease];
picker.delegate                          = self;
picker.allowsPickingMultipleItems   = YES;
picker.prompt                            = @"Choose Media To Play";

[self presentModalViewController:picker animated:YES];
```

MPMusicPlayerController

`allowsPickingMultipleItems = YES`



`allowsPickingMultipleItems = NO`

MPMusicPlayerController

- The MPMediaPickerController tells its delegate when music is selected, and returns the item collection

```
- (void)mediaPicker:(MPMediaPickerController*)mediaPicker
  didPickMediaItems:(MPMediaItemCollection*)mediaItemCollection {

  /* Update music player */
  [musicController setQueueWithItemCollection:mediaItemCollection];

  /* Dismiss picker */
  [self dismissModalViewControllerAnimated:YES];
}

- (void)mediaPickerDidCancel:(MPMediaPickerController *)mediaPicker {
  [self dismissModalViewControllerAnimated:YES];
}
```

MPMusicPlayerController

- The MPMusicPlayerController uses notifications to indicate when certain events occur (track finished, volume changed, etc)

```
[musicController beginGeneratingPlaybackNotificaitons];
[musicController endGeneratingPlaybackNotificaitons];

MPMusicPlayerControllerPlaybackStateDidChangeNotification
MPMusicPlayerControllerNowPlayingItemDidChangeNotification
MPMusicPlayerControllerVolumeDidChangeNotification
```

MPMusicPlayerController

- So register your view controller to handle those notifications and update the UI, e.g.

```
NSNotificationCenter *notificationCenter =
                [NSNotificationCenter defaultCenter];

[notificationCenter
 addObserver:self
 selector:@selector(handleNowPlayingItemChanged:)
 name:MPMusicPlayerControllerNowPlayingItemDidChangeNotification
 object:musicController
];
```