ES3 Lab 1

Beginning iPhone development

Resources

- The image files used in this lab are at
  - http://www.dcs.gla.ac.uk/~jhw/es3/lab1.zip

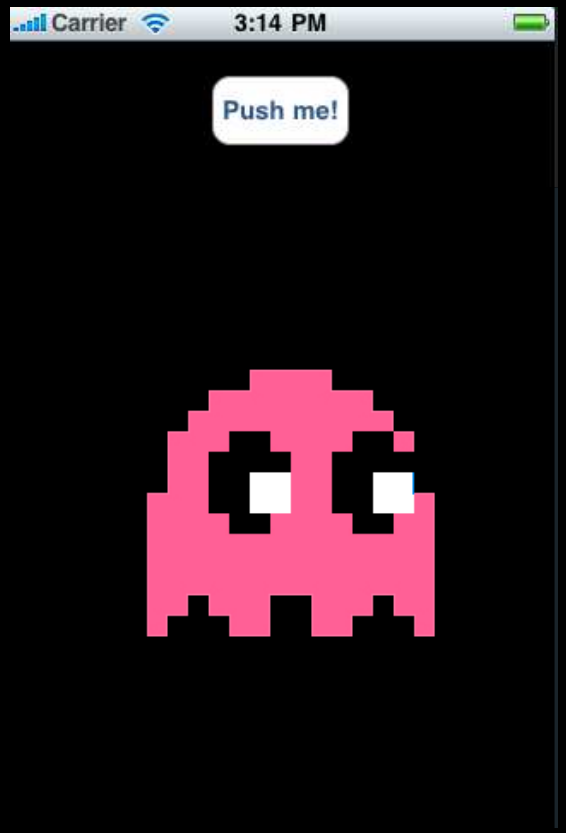
This lab

- XCode:
  - Learn how to create an iPhone project
  - Build and run on the simulator
  - Understand the files that will be created for you
  - Adding frameworks
  - Adding resources
  - Setting the icon, and start-up image

- Development:
  - Add some simple controls programatically
  - Display an image
  - Use basic animation
  - Custom view drawing

Exercise

- Push a button, make the sprite move to positions loaded from a file (and play sounds while it does)



Installing XCode

- Lab machine have this, but if you're on your own laptop...
  - Need OS X 10.6 (Snow Leopard)
  - Need to sign up for a (free) developer account at developer.apple.com
  - Then go to developer.apple.com/iphone and download iPhone SDK 3.1.2 (as of December 2009)
  - Download the (2.2Gb!) disk image and install it
- That's it: everything, including the IDE, simulator and all other tools are installed for you

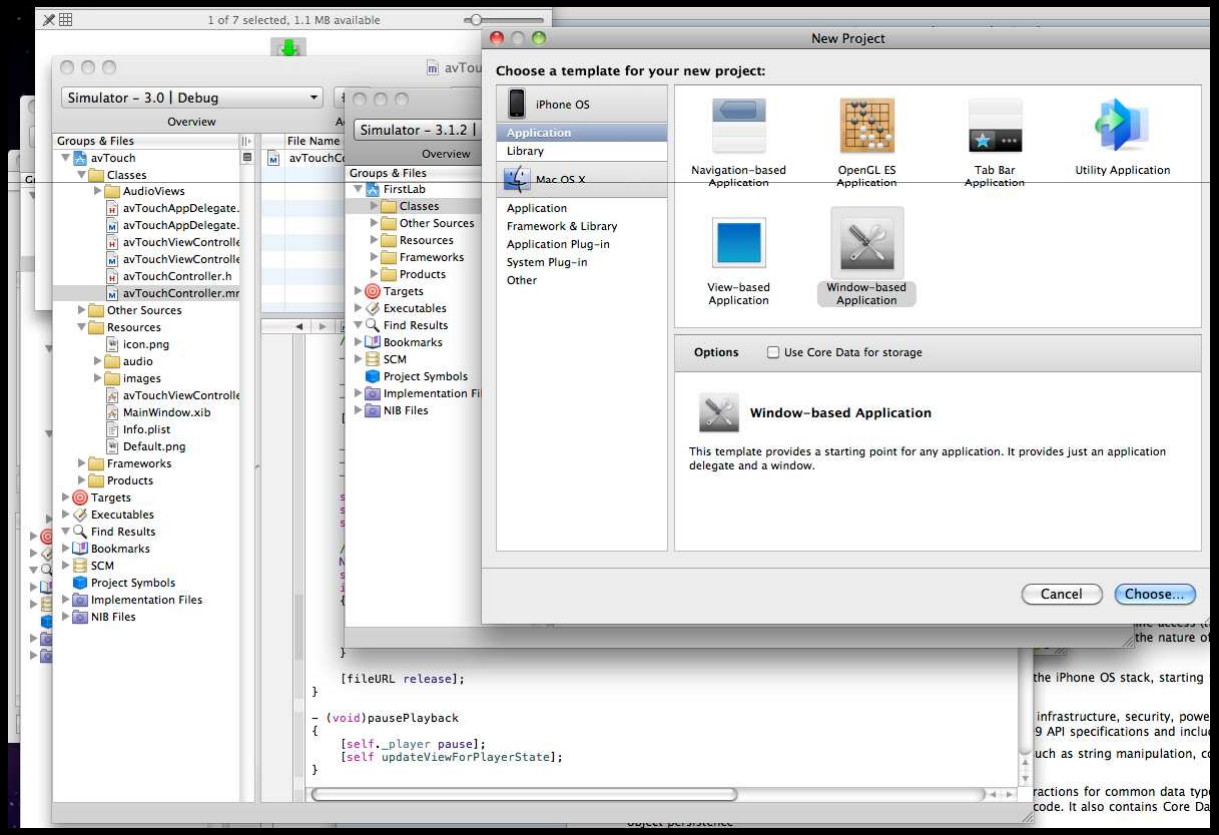
XCode tips

- Use autocomplete (tab to complete)
- If you press], XCode will insert a matching [in the right place

- **Alt-Cmd-Up** to switch between .h and .m files
- **Alt-Dbl-click** on a name to look it up in the API
- **Shift-Cmd-B** Build the project
- **Alt-Cmd-Enter** Run the project
- **Shift-Cmd-R** show the console

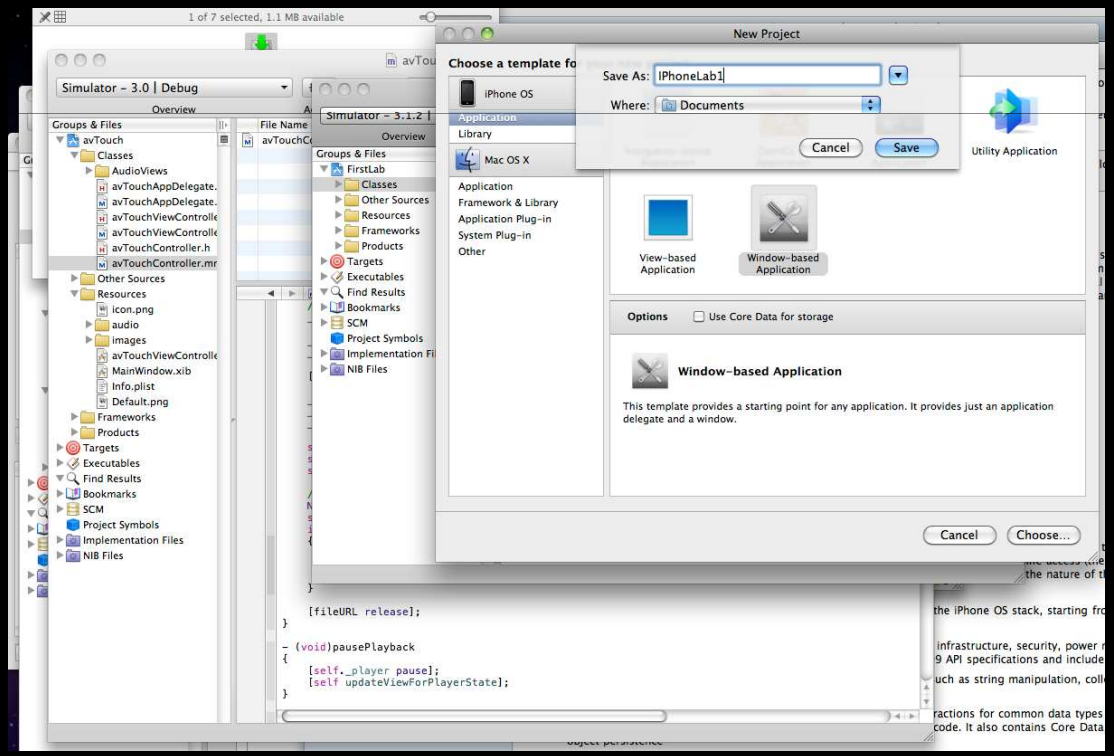
Creating a project

- Start XCode, then File/New project...
  - Create a Window-based project

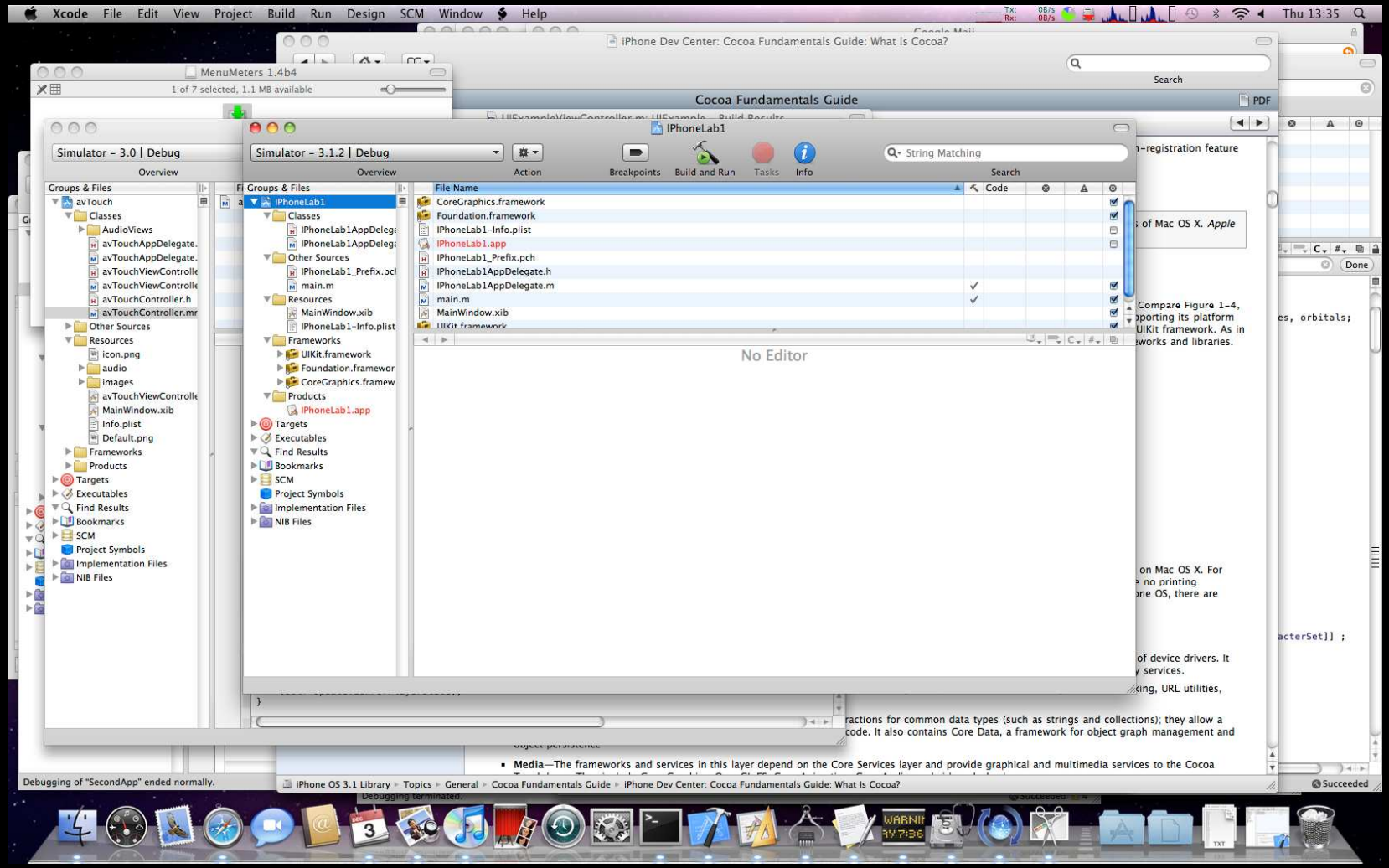


Creating a project

- Give it a name and a project will be created. XCode should look like below



XCode layout



main.m

- This contains the main function which is the entry point for an Objective-C application.
- In general, you never need to use this. Just starts UIApplication
  - UIApplication is linked to AppDelegate via InterfaceBuilder (magic for just now)
- Note that this is where the autorelease pool is created for memory management...

Xcode File Edit View Project Build Run Design SCM Window Help

iPhone Dev Center: Cocoa Fundamentals Guide: What Is Cocoa?

MenuMeters 1.4b4

1 of 7 selected, 1.1 MB available

Cocoa Fundamentals Guide

Simulator - 3.0 | Debug

Overview

Groups & Files

- avTouch
  - Classes
    - AudioViews
    - avTouchAppDelegate.
    - avTouchAppDelegate.
    - avTouchViewControlle
    - avTouchViewControlle
    - avTouchController.h
    - avTouchController.mr
  - Other Sources
  - Resources
    - icon.png
    - audio
    - images
    - avTouchViewControlle
    - MainWindow.xib
    - Info.plist
    - Default.png
  - Frameworks
  - Products
- Targets
- Executables
- Find Results
- Bookmarks
- SCM
- Project Symbols
- Implementation Files
- NIB Files

main.m - IPhoneLab1

Simulator - 3.1.2 | Debug

Overview Action Breakpoints Build and Run Tasks Info Search

String Matching

Groups & Files

- IPhoneLab1
  - Classes
    - IPhoneLab1AppDeleg
    - IPhoneLab1AppDeleg
  - Other Sources
    - IPhoneLab1_Prefix.pcl
    - main.m
  - Resources
    - MainWindow.xib
    - IPhoneLab1-Info.plist
  - Frameworks
    - UIKit.framework
    - Foundation.framewor
    - CoreGraphics.framew
  - Products
    - IPhoneLab1.app
- Targets
- Executables
- Find Results
- Bookmarks
- SCM
- Project Symbols
- Implementation Files
- NIB Files

File Name	Code
main.m	✓

main.m:1 <No selected symbol>

```
//
//  main.m
//  IPhoneLab1
//
//  Created by John Williamson on 03/12/2009.
//  Copyright __MyCompanyName__ 2009. All rights reserved.
//

#import <UIKit/UIKit.h>

int main(int argc, char *argv[]) {

    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    int retVal = UIApplicationMain(argc, argv, nil, nil);
    [pool release];
    return retVal;
}
```

n-registration feature

of Mac OS X. *Apple*

Compare Figure 1-4,
pporting its platform
UIKit framework. As in
works and libraries.

on Mac OS X. For
e no printing
one OS, there are

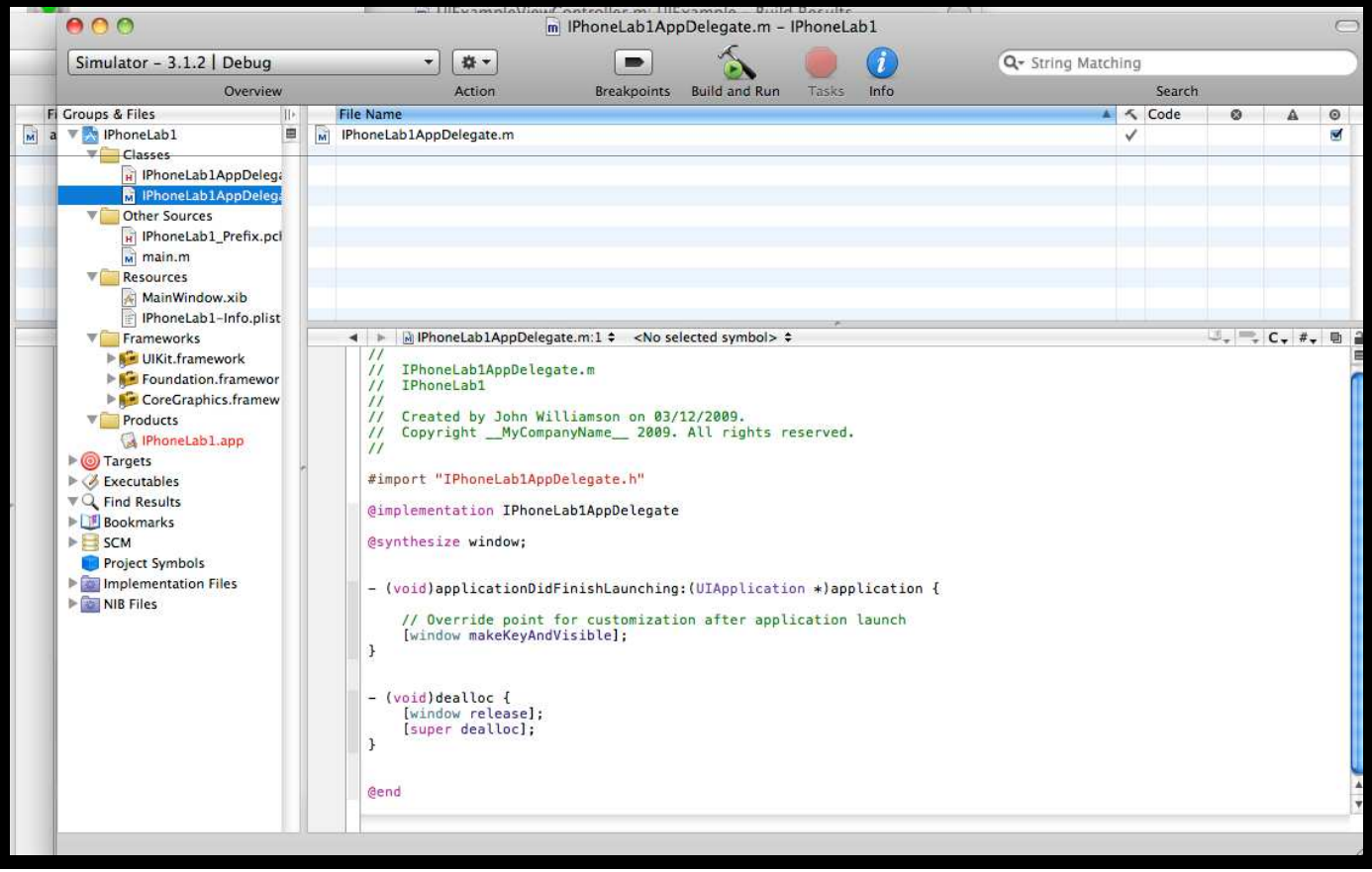
of device drivers. It
y services.

king, URL utilities,

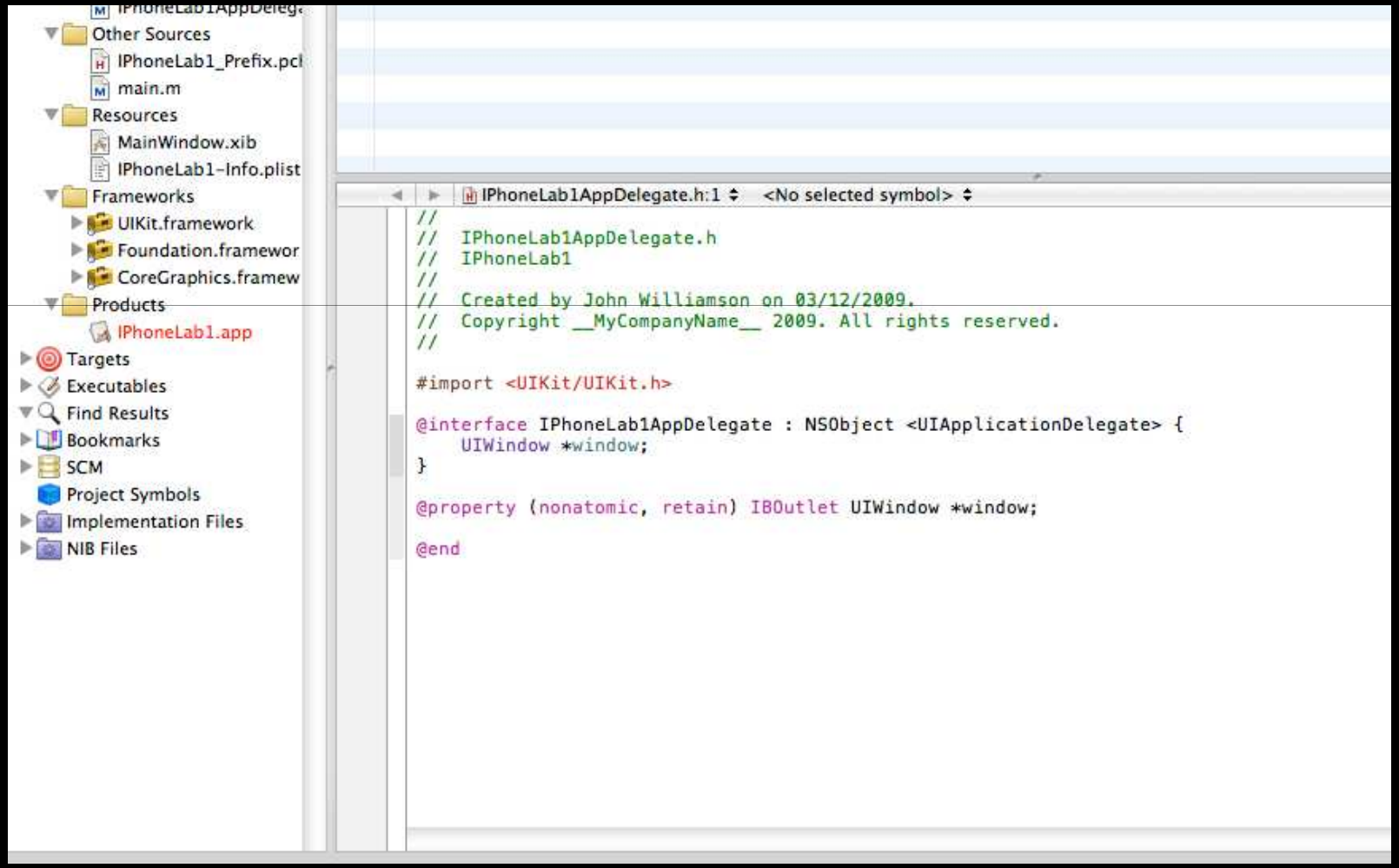
ractions for common data types (such as strings and collections); they allow a
code. It also contains Core Data, a framework for object graph management and

iPhoneLab1AppDelegate.m

- Main application class
  - Everything is launched from here, in applicationDidFinishLaunching



iPhoneLab1AppDelegate.h



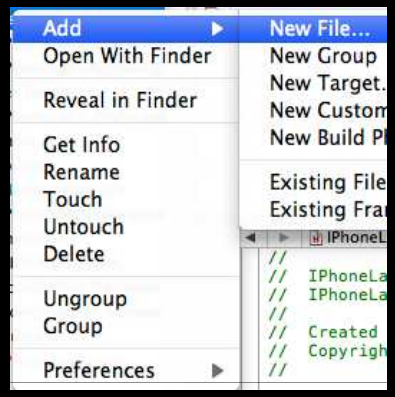
A tip: NSLog

- Use NSLog to write things to the console (Run/Console to show the console)
- Works like printf
  - Note that NSString's are written out using the %@ specifier, not %s

```
if(badErrorCode)
          NSLog(@"Something really bad happened in doSomething!\n");

//Using the format
int i;
i = 8;
NSLog(@"i=%d\n", i);
```

Creating a UI



- Add a new class (right-click classes/Add new file...)
  - Make it a subclass of UIViewController
  - Call it SimpleView

- In Lab1AppDelegate.h, add a SimpleView instance variable

```
SimpleView *viewController;
```

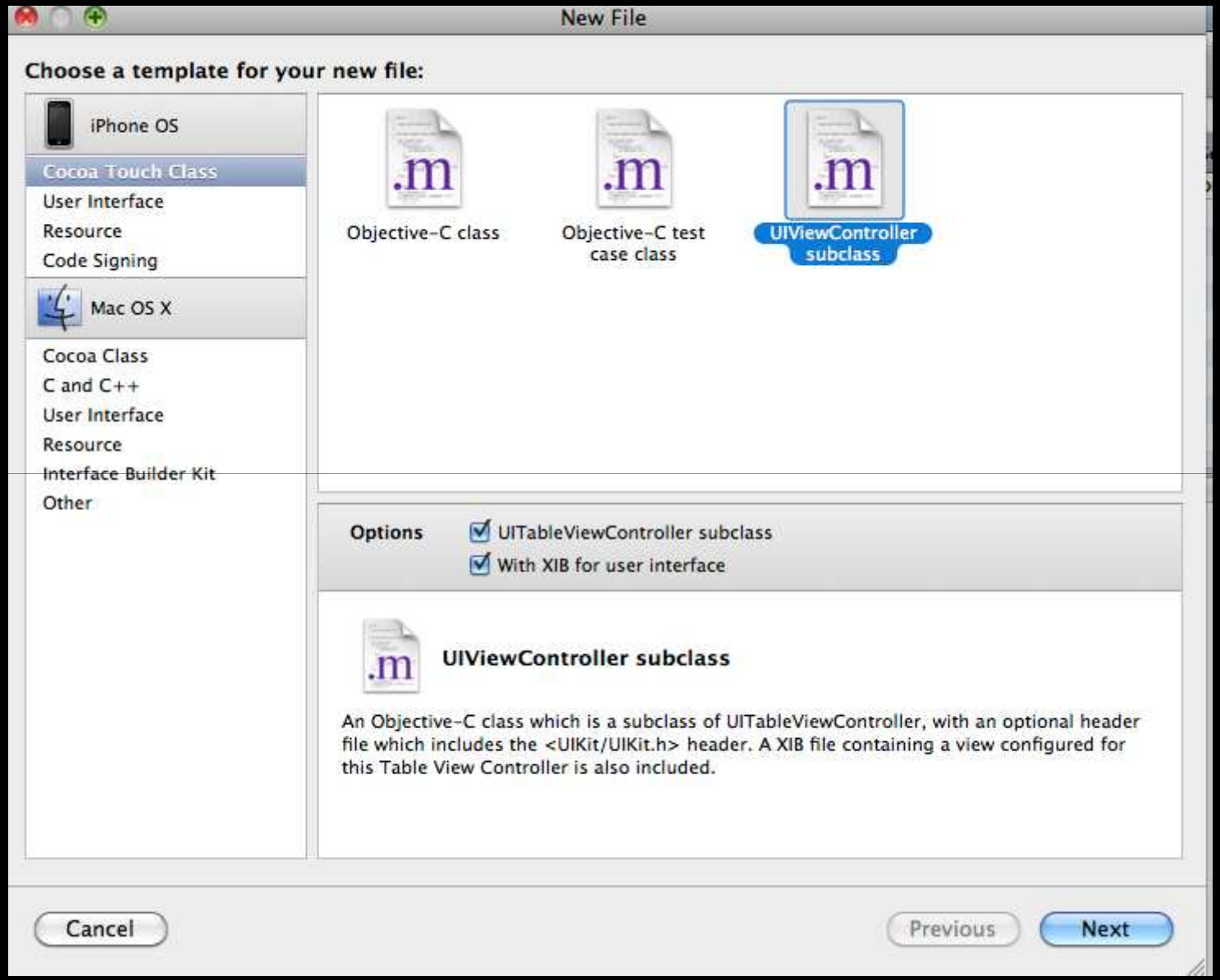
- Add a property for it. Make it (nonatomic, retain)

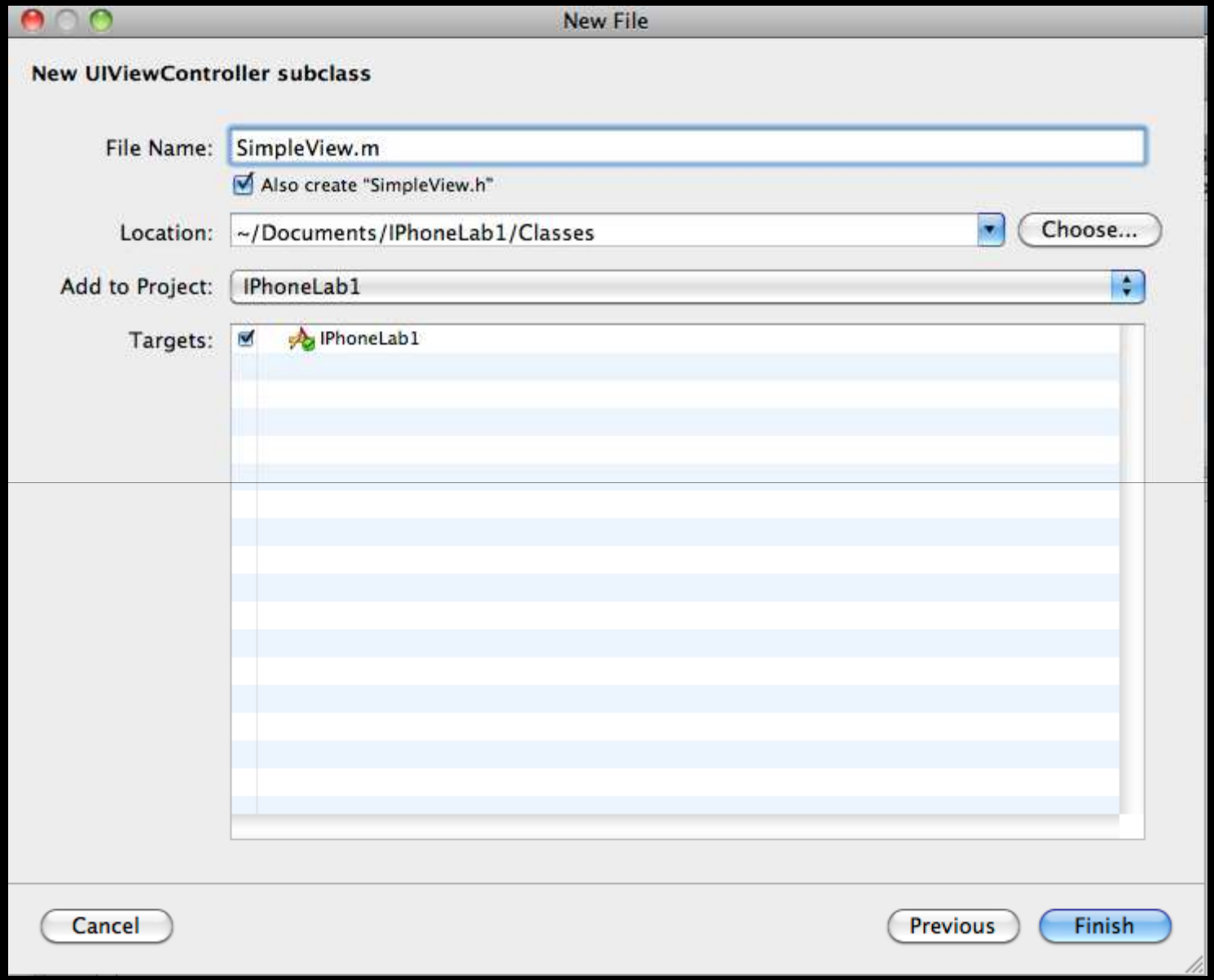
```
@property (nonatomic, retain) SimpleView *viewController;
```

  - Remember, properties go after the instance variable block!

  - To finish adding the property, add @synthesize viewController immediately after @implementation IphoneLab1

```
@implementation IphoneLab1
@synthesize viewController;
```


New File
Choose a template for your new file:
iPhone OS
Cocoa Touch Class
User Interface
Resource
Code Signing
Mac OS X
Cocoa Class
C and C++
User Interface
Resource
Interface Builder Kit
Other
Objective-C class
Objective-C test case class
UIViewController subclass
Options
UITableViewController subclass
With XIB for user interface
UIViewController subclass
An Objective-C class which is a subclass of UITableViewController, with an optional header file which includes the <UIKit/UIKit.h> header. A XIB file containing a view configured for this Table View Controller is also included.
Cancel
Previous
Next


New File
New UIViewController subclass
File Name:
SimpleView.m
Also create "SimpleView.h"
Location:
~/Documents/IPhoneLab1/Classes
Choose...
Add to Project:
IPhoneLab1
Targets:
IPhoneLab1
Cancel
Previous
Finish

Creating a UI

- In Lab1AppDelegate.m instantiate the SimpleView and add it's view to the window
  - Put it in applicationDidFinishLaunching, before [**window makeKeyAndVisible**];

```
self.viewController = [SimpleView alloc];
[self.window addSubView:self.viewController.view];
```

- Note that you add the **view** member to the window, not the UIViewController class itself
- The **window** object is the top level container for all UI components
- The use of **self** in the assignment is ESSENTIAL
  - otherwise the property won't be used, and the retain mechanism won't work!
- **appDidFinishLaunching** is the main entry point for your code

Adding some components

- To make a UI, first of all create a view -- a **UIView** instance
  - Like most UI components, UIView is initialized with a frame
  - Rectangle which specifies size and position of component
  - The **CGRectMake(x,y,width,height)** function creates such rectangles!

- We're doing programmatic UI creation, so we need to add the creation code to **loadView** in **SimpleView.m .** This is called when the view is created.

```
- (void)loadView {
          self.view = [[UIView alloc] initWithFrame:CGRectMake(0,0,320,480)];
          self.view.backgroundColor = [UIColor redColor];
}
```

- Build this and run it
  - If you don't get a red screen, something went wrong

```
// Implement loadView to create a view hierarchy programmatically, without using a nib.
- (void)loadView {
    //Create the view
    self.view = [[UIView alloc] initWithFrame:CGRectMake(0, 0, 320, 480)];


}
```

```
//  IPhoneLab1AppDelegate.m
//  IPhoneLab1
//
//  Created by John Williamson on 03/12/2009.
//  Copyright __MyCompanyName__ 2009. All rights reserved.
//

#import "IPhoneLab1AppDelegate.h"
#import "SimpleView.h"

@implementation IPhoneLab1AppDelegate

@synthesize window, simpleViewController;


- (void)applicationDidFinishLaunching:(UIApplication *)application {


    self.simpleViewController = [SimpleView alloc]; // must release this later!
    [self.window addSubview:self.simpleViewController.view]; // add the view to the window

    // Override point for customization after application launch
    [window makeKeyAndVisible];
}


- (void)dealloc {
    [simpleViewController release]; // matches the alloc in init
    [window release];
    [super dealloc];
```

```
//
//  IPhoneLab1AppDelegate.h
//  IPhoneLab1
//
//  Created by John Williamson on 03/12/2009.
//  Copyright __MyCompanyName__ 2009. All rights reserved.
//

#import <UIKit/UIKit.h>

@class SimpleView;

@interface IPhoneLab1AppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow *window;
    SimpleView *simpleViewController;
}

@property (nonatomic, retain) IBOutlet UIWindow *window;
@property (nonatomic, retain) SimpleView* simpleViewController;
@end
```

Task: Add a button

- The UIButton class is a simple push button
- Add this to SimpleView
  - Remember to create an instance variable and property, synthesize it, allocate and initialize it in loadView
    - To create a UIButton, use [**UIButton buttonWithType:UIButtonTypeRoundedRect**]
    - Don't use the alloc / initWithFrame method
    - Set the frame instance variable afterwards

      ```
      mybutton.frame = CGRectMake(...
      ```

  - Set the title using **UIButton:setTitle:withType**
    - Use **UIControlStateNormal for** the type

- Now, also change the **SimpleView** background to black! (red is rather ugly)

Adding a resource

- Add files to your project by right clicking the resource tab in the left hand pane

- Add pacman.png from the resource zip
  - Just click through the dialogs -- the defaults will work

Drawing the image

- To draw the image, we need to load it and draw it
- We'll make a custom UI component which displays an image
- Add a new class ImageView
  - Choose "Subclass of UIView" in the dropdown menu

Task: Load the image and display it

- Use an instance of **UIImage** to load the image
- Add a new method to **ImageView** -- **initWithImage:at**:, taking a filename and a position
  - assume .png extension for the filename
  - return a value of type **ImageView** * -- i.e. **self**!
- Use **NSBundle** methods to get the path to the resource
  - Remember, you can get the main bundle with [**NSBundle mainBundle**]
- Load the image into an instance variable of **UIImage*** in **ImageView**
  - Remember to make and synthesize a property
- Then, at the end of **initWithImage** method, call [self **initWithFrame:frame**], calculating the rectangle for the image given the image size and the position specified

Drawing the image

- Add the following to the method **drawRect**

```
- (void) drawRect:(CGRect)rect
{
   [self.viewImage drawInRect:rect]; // use your image member in place of viewImage

}
```

- This just draws the image on the control

Task: Load the image and display it

- In SimpleView, add an instance variable and property for a ImageView* object
  - Remember to **#import "imageview.h"** in SimpleView.h
- In loadView, instantiate it with

```
//Use the pacman image
self.imageView = [ImageView initWithImage:@"pacman" at:CGPointMake(100,50)];
```

- Add it as a subview, in the same way as the button was added
- Build it, and check that it appears on screen

Responding to a button push

- Need to add a target/action for the button
  - do this in **loadView** of **SimpleView**
  - make it send a message to the **SimpleView** instance when the finger comes up over the button

```
[pushButton addTarget:self action:@selector(buttonPushed:)
forControlEvents:UIControlEventTouchUpInside];
```

- This will send a **buttonPushed** message to the current instance of **SimpleView**
  - Add a blank method with this name to the prototype and body of the class (.h and .m files)

```
- (void) buttonPushed:(id)sender;
```

  - Note that it must take one parameter of type **id** -- the object that sent the message
  - If you want, add a logging command with **NSLog** and check the method is called when you press the button

Task: Playing a Sound

- Add the AudioToolbox framework to your app (right click frameworks, choose **Add /Existing framework...)**
- Add **#import <AudioToolbox/AudioServices.h>** to the top of SimpleView.h
- Add **bleep_sound.wav** to the resources
- Add a variable of type SystemSoundID to SimpleView.h and a property for it (remember to synthesize it!). Don't make it a **retain** property - it's not an Objective C object!

```
SystemSoundID bleepSound;
```

- To play a sound, you must first get a reference to it
- AudioToolbox is a C-framework, so you must convert URLs to CFURLs

```
NSString *bleepSoundPath = [[NSBundle mainBundle] pathForResource:@"bleep_sound"
ofType:@"wav"];
CFURLRef bleepSoundURL = (CFURLRef) [NSURL fileURLWithPath:bleepSoundPath];
AudioServicesCreateSystemSoundID(bleepSoundURL, &bleepSound); // note the use of &
```

Playing the Sound

- It's easy to play the sound
  `AudioServicesPlaySystemSound(bleepSound);`
  - Add this to the code that responds to the button press, so that the sound plays in sync with the animation
  - Build it, and check that the sound plays correctly
- Note: there are many iPhone audio API's, which are much more powerful than AudioToolbox (just plays short wave files)
  - Especially OpenAL, gives flexible multichannel audio playback with spatialization, streaming etc.

Making an animation

- Use the iPhone's simple and powerful animation effect
- This involves basically just setting up a transformation, and telling the iPhone to start it up

- We'll make our sprite do a spinning zoom out and fade out, before reversing back in again
  - Sounds hard, but it's just a few lines of code!

- Create a **doSpin** method in **SimpleView**

Animations

- You create an animation by placing changes to be displayed between a [**UIView beginAnimation**] and [**UIView commitAnimation**] (which will start the animation sequence)
  - All you need to do is specify how things should look at the end -- Cocoa will do the actual animation

```
[UIView beginAnimatons:@"spin" context:nil];
[UIView commitAnimations];
```

- Add this to **doSpin**, and add a call to **doSpin** from **buttonPushed**
- **Nothing will happen!**

Animations

- You need to specify at least how long the animation will last, and some changes to make
  - Add the following

```
[UIView beginAnimatons:@"spin" context:nil];
[UIView setAnimationDuration:0.5]; // 0.5 seconds

[UIView setAnimationBeginsFromCurrentState:YES];
// this is just to make it start from where we are

[UIView setAnimationRepeatAutoreverses:YES];
// make it reverse automatically

[UIView setAnimationRepeatCount:1];
//Just play it once...

//Now make it fade out
self.imageView.alpha = 0;

[UIView commitAnimations];
```

Test it!

- If you press the button now, the sprite should fade out, then in again
  - Except it disappears at the end...
  - ...because we set the alpha to zero
  - We need to reset after the animation has finished

- This is easy -- animations can send messages when they stop
  - Add this before **commitAnimation**

```
[UIView setAnimationDidStopSelector:@selector(animationStopped:)];
[UIView setAnimationDelegate:self];
```

- This will send an **animationStopped** message to self when it finishes
  - Create an **animationStopped** method in SimpleView
  - In it, just set the alpha of imageView to 255 (fully opaque)

Make it spin and zoom!

- Now, if you test it, the sprite should smoothly return to it's original state

- We can add one last effect -- spin and zoom
  - Changes to position, size and rotation are made by changing the **transform** property of a control
  - The **CGAffineTransformMake*** methods make it easy to create such transforms

- Add this before **commitAnimation**

```
CGAffineTransform spin = CGAffineTransformMakeRotation(360); // spin 360 degrees
CGAffineTransform zoom = CGAffineTransformMakeScale(10,10);  // scale 10 times
CGAffineTransform spinZoom = CGAffineTransformConcat(spin, zoom); // join them
self.imageView.transform = spinZoom;
```

- Note that you only specify the endpoint of the animation -- nothing about how it will execute
  - Makes it very easy to use

Now try it...

- And note that the same problem occurs with the state sticking
- Add a line to animationStopped to set the imageView's transform to **CGAffineTransformIdentity**
  - This is the no transformation state

Polish: start up image and icon

- If you add a 320x480 image called **Default.png** to the resources, it will be shown as app loads
  - Increases apparent load speed -- all apps should use this according to Apple guidelines.
- Similarly, add a 57x57 PNG image called **Icon.png**, and it will become the icon
  - Note: you don't include the gloss/rounded corner effect -- the iPhone does this for you
- **Add the provided Default.png and Icon.png to the resources**
  - **They will automatically be used by the app**
- Build, and check that the icon is right, and that the background appears as it loads

Result