Lab Project

Building a Simple App from Scratch
using Xcode 4.5

Air Horn

Air Horn is a simple application by Alex Miyamura. It displays a compressed air horn on the screen. When a user taps the image, the phone plays an air horn sound.

Alex reported that it took him about 3-5 hours to write.

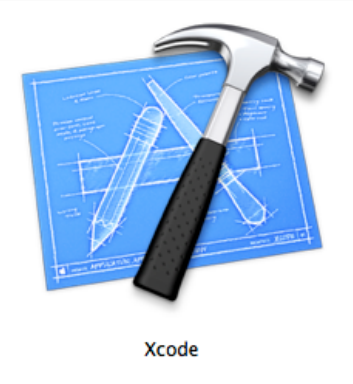
Air horn went to #1 on the App Store, with millions of downloads. The app is free, but it has brought in over $100,000 in advertising revenue for Alex.

Important lesson: Apps don't have to be complicated or fancy in order to be successful. Simple apps can do quite well.

Let's Create a Horn App

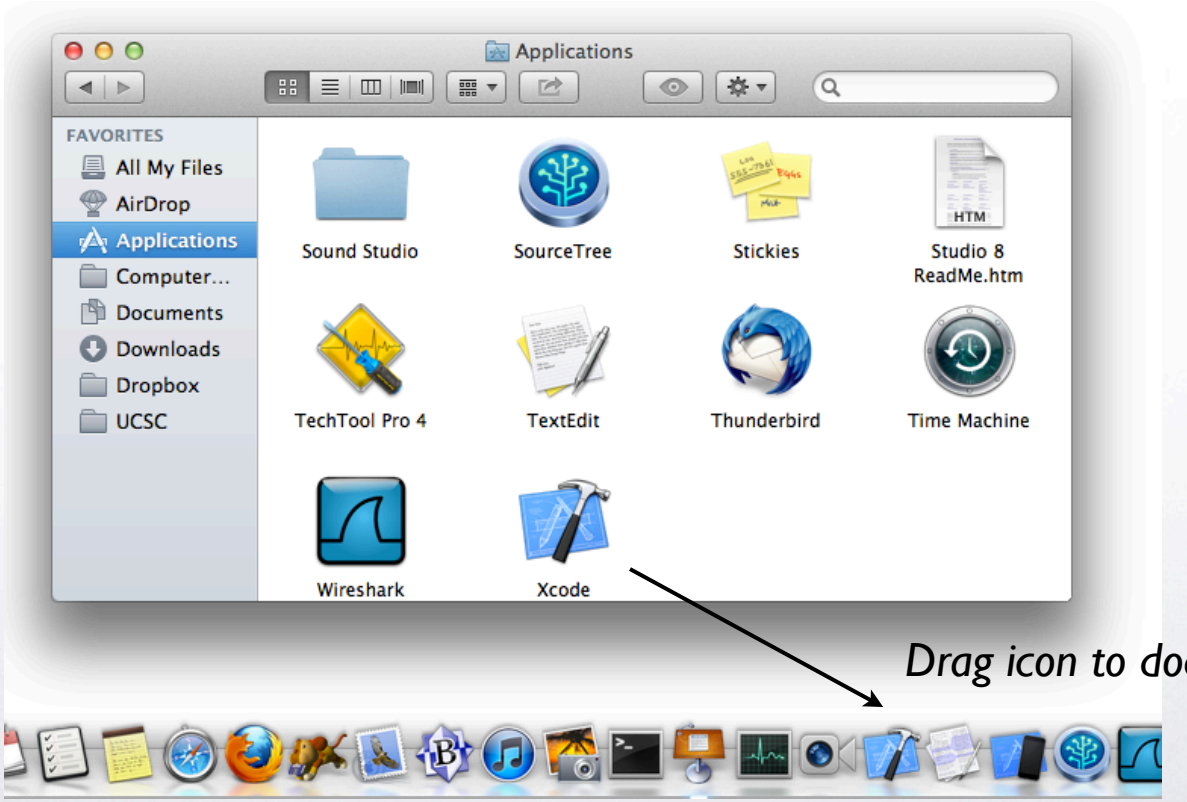
Many developers have created apps similar to Alex's Air Horn. Let's see if we can do the same. We'll create an app called Air Euphonium.

Let's start by launching Xcode. It's probably in your Mac's icon doc at the bottom or side of your screen. If not, we can find it in the Applications folder on our Mac hard disk.



Adding Xcode to the Dock

If Xcode isn't in your Mac's dock, drag it there. That will make it easier for us to find it in the future. We're going to be using it a lot!

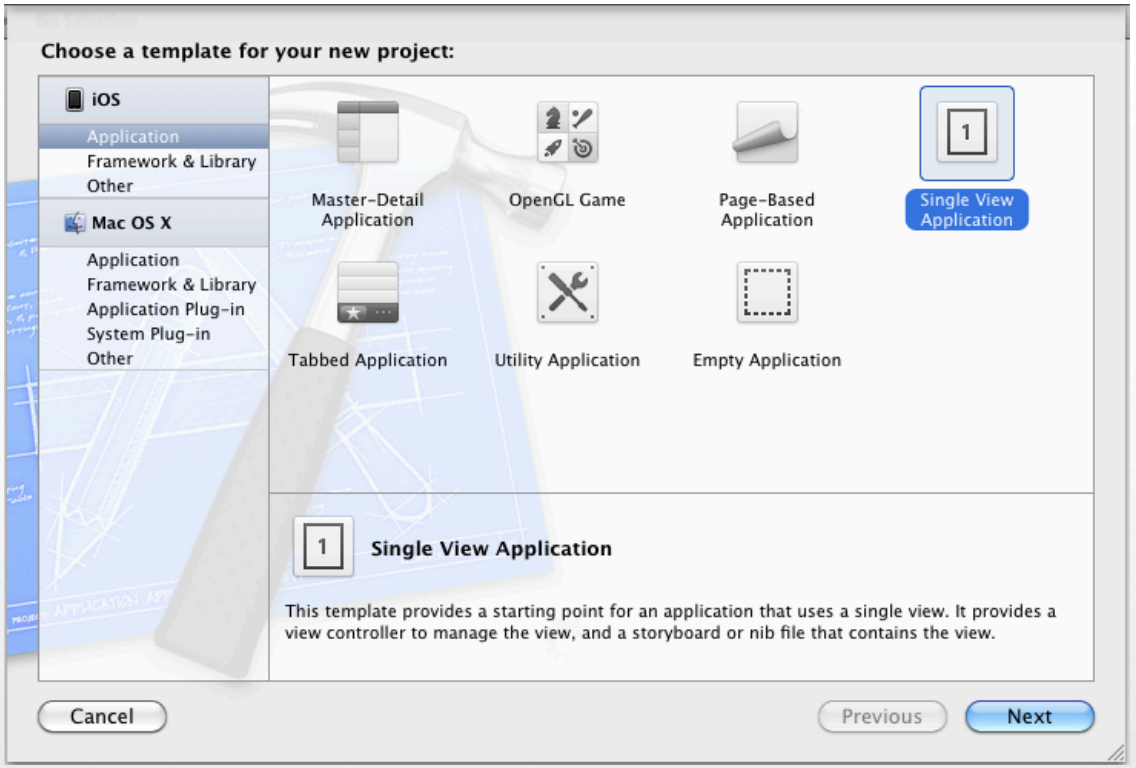


Creating a New Project

Double Click on Xcode to open it, then select "Create a New Xcode Project."

Selecting our Project Type

Now take a look at the different kinds of projects we can make. We're going to make a Single View Application for iOS. So select that option, then click on the "Next" button.

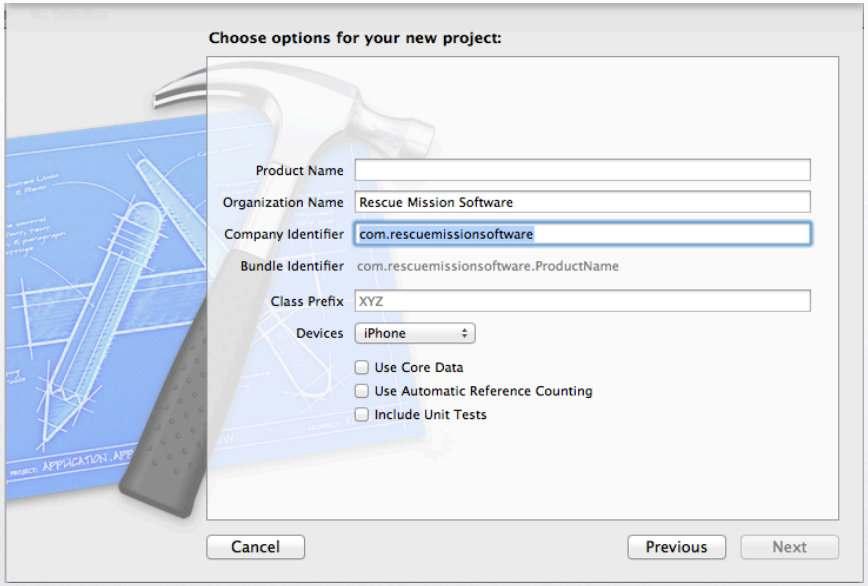


Selecting our Project Options

Next Xcode is going to ask us for our Product Name and Company Identifier. For Product Name, enter: AirEuphonium[YourNameHere]

When naming projects for class, always include your own name in the name of the project. With a large number of students, that will make it easier for us to tell to whom each project belongs.

For company identifier, enter: edu.ucsc



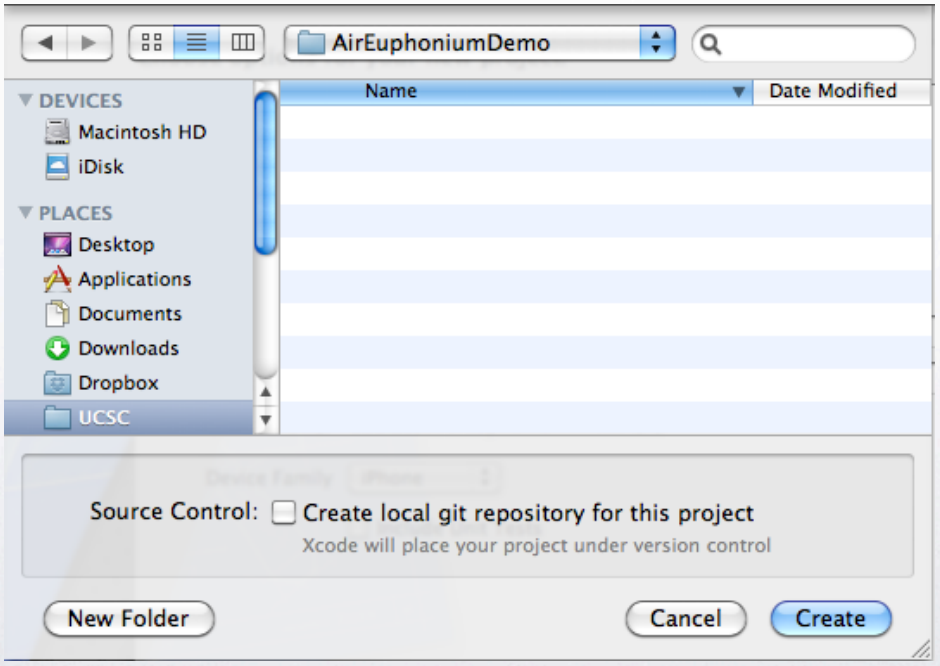
Choose iPhone for the Device Family.

For this project, uncheck the options for Use Storyboard, Use Automatic Reference Counting, and Include Unit Tests.

Choosing the Location of your Project Files

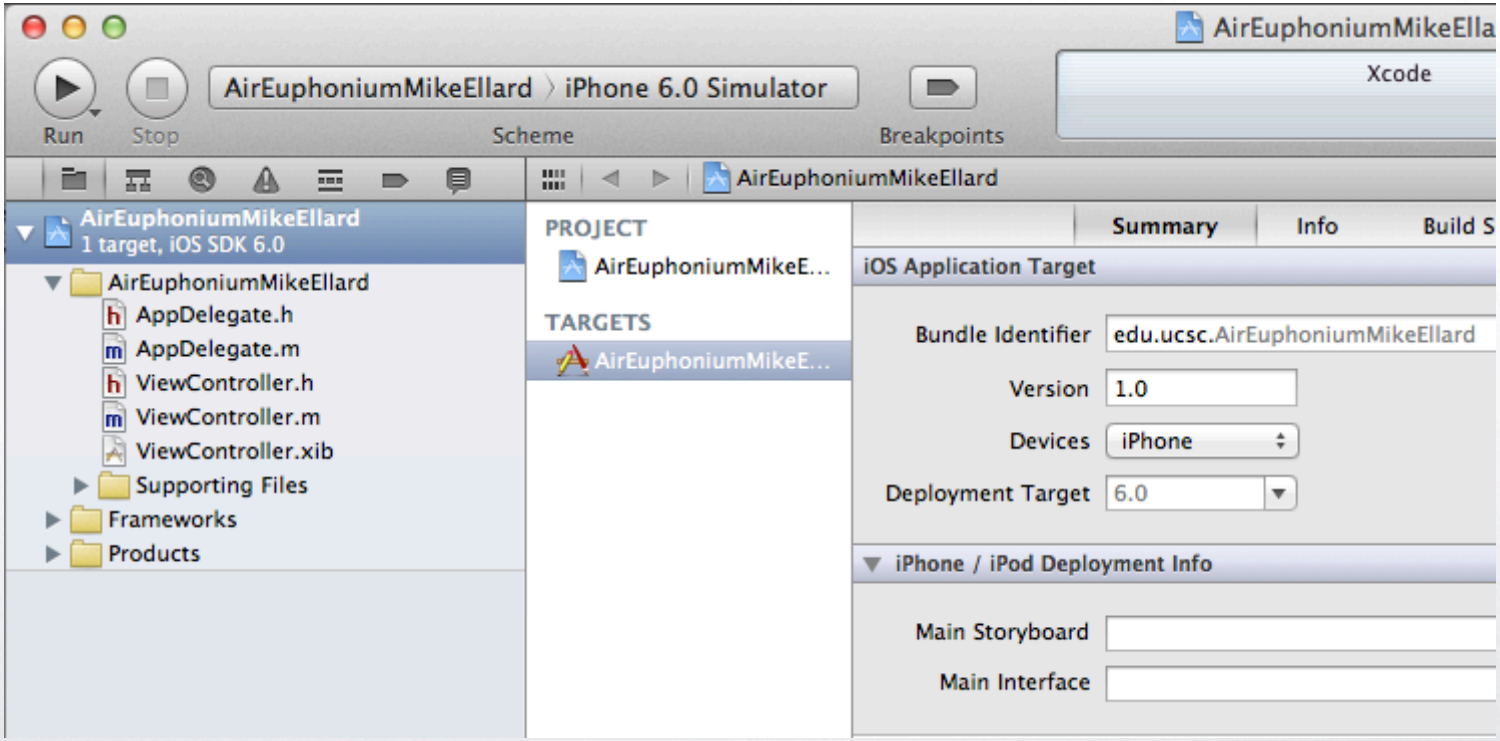
Next, Xcode is going to ask us where to put our project files. Pick an appropriate place on your hard disk or thumb drive.

You can leave the "Local git repository" checkbox unchecked. We're not going to be using that feature for this project.



A first look at your Project

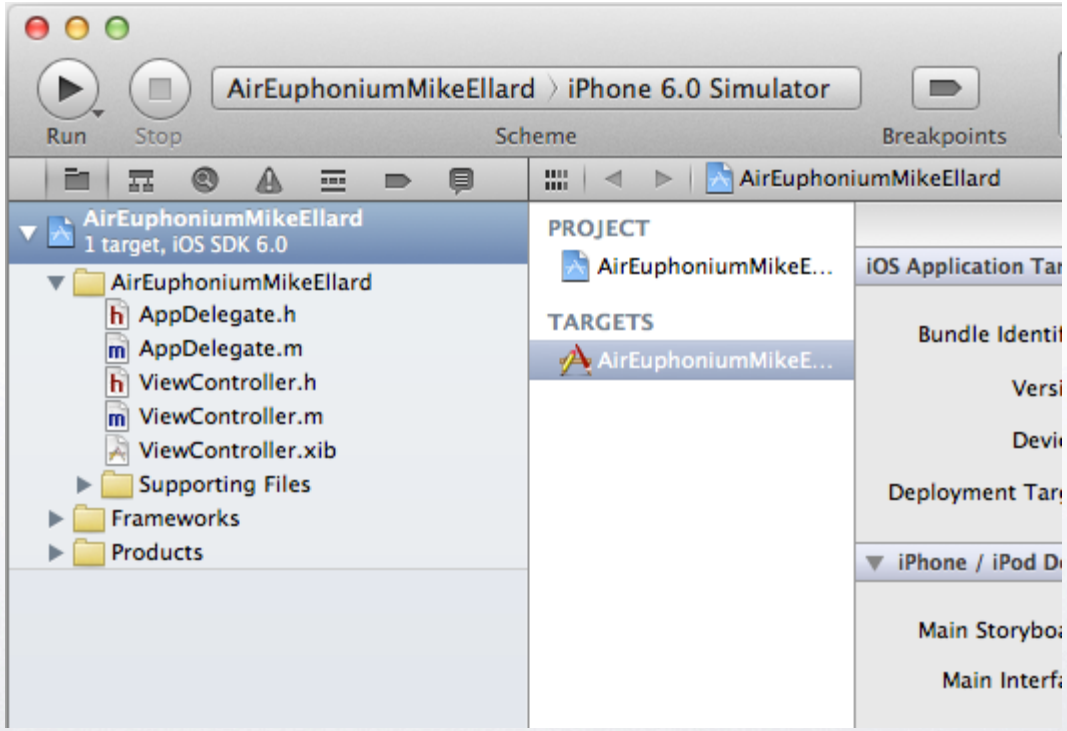
We haven't done anything, and there are already a lot of files there. Great! Xcode has already done a lot of the work for us. We're going to ignore most of what we see for now - we'll go into more detail in a later class.



Building and Running Your Project

Next, click the Run button. Your project will compile and launch in the iOS Simulator.

Click this Button



A first look at our Project

Looking at our project in the Simulator, it's pretty boring at this point.

It doesn't even say, "Hello World!"

Having verified that we have a working app, let's go back to Xcode so we can make it do something.

When you first launch your app in the Simulator, the Simulator's screen may be black for a little while as the app loads. Be patient - the gray screen is coming!

Returning to Xcode

There are two different ways we can get back to Xcode.

- We can click on the Xcode icon in the dock.
- We can click in any active Xcode window.

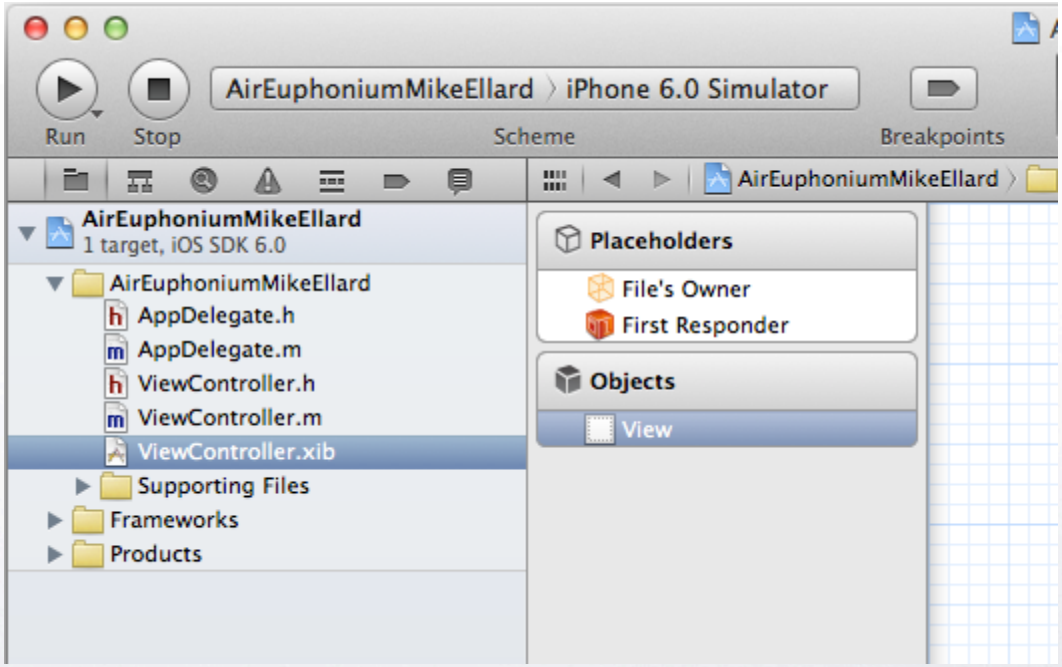
You can choose whichever method is easier for you.

Note: On some machines, you may notice that the Xcode icon shows up in the dock when Xcode is running, but not otherwise. This is a standard Mac behavior for applications that haven't been permanently added to the dock.

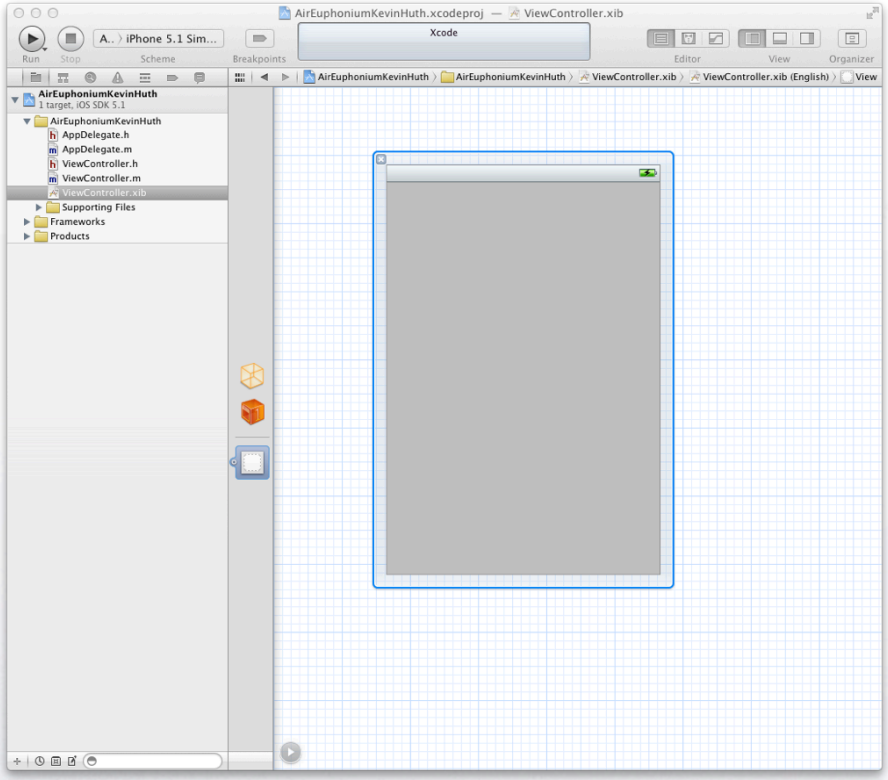
Working with a XIB file

Back in Xcode, click on the file called ViewController.xib.

Click on this file



A First Look at our XIB



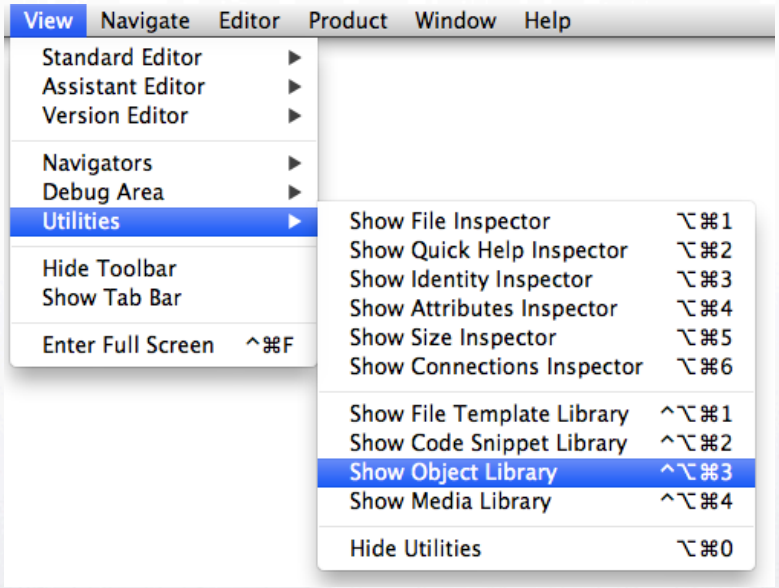
Xcode's Interface Builder component will show us the XIB file's main UIView.

It's not very interesting right now, because our main UIView doesn't contain any other objects, but we're going to change that in the slides that follow.

Displaying the Library

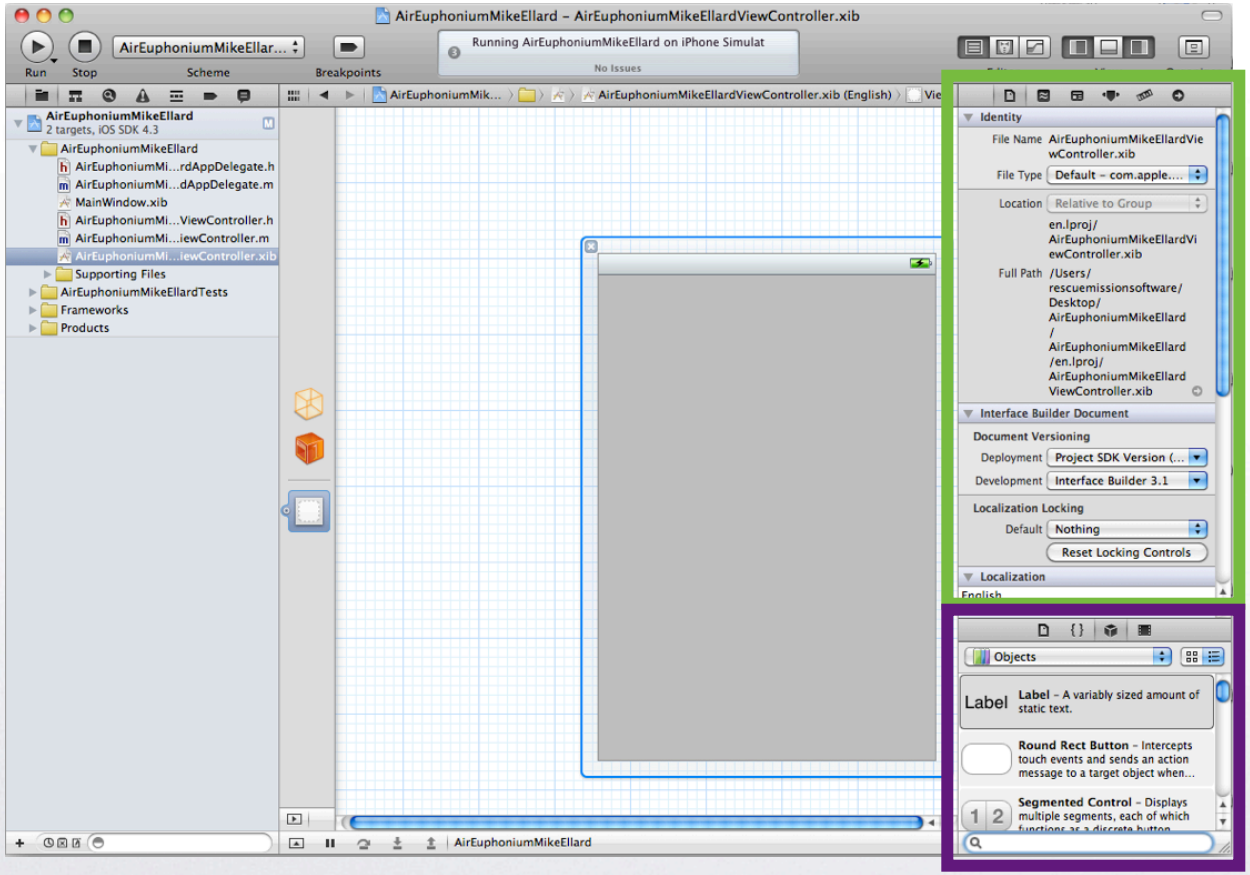
To add objects to our view, we're going to need to display the Object Library.

To make the Object Library visible, choose the "Object Library" command from the "Utilities" submenu of Xcode's "View" menu.



Our First Look at the Library

The library pane shares the right side of your screen with the inspector pane.

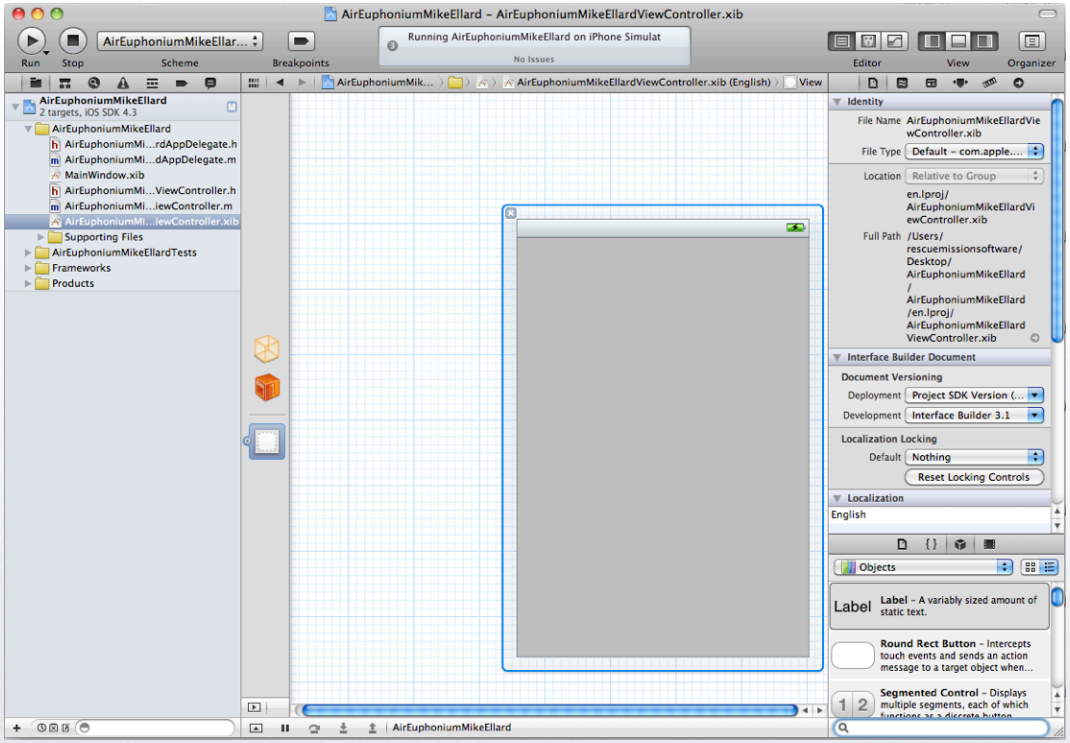


The Inspector Pane

The Library Pane

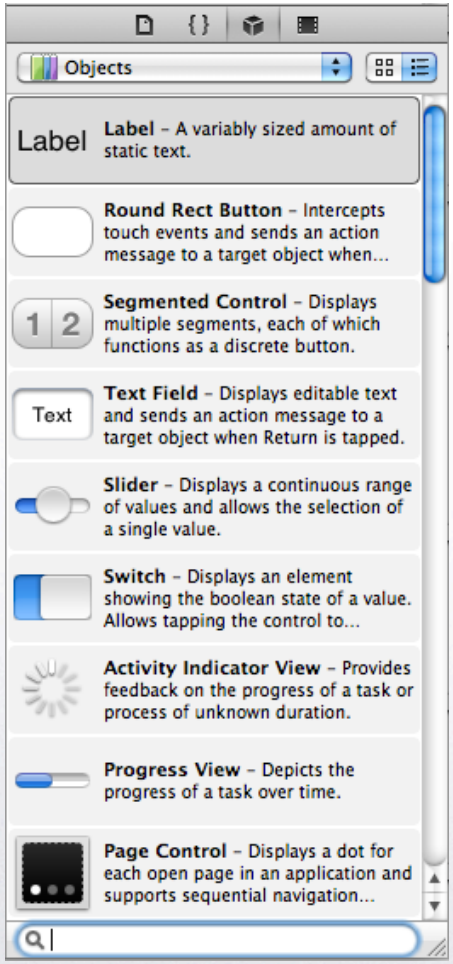
Enlarging the Library Pane

If you are working on a machine with a small screen, you may find that the library is pretty tiny. You can make the library pane larger by clicking on the divider between the library pane and the attributes pane and dragging upwards.



Click and drag up here to make the Library Pane larger.

Exploring the Library



In the library, you will see a wide variety of user interface objects that you can configure using Interface Builder.

Take a moment to explore the different kinds of objects you could use.

First Steps in Building our Interface

Let's start building the interface for our project. First, drag a Round Rect Button from the Library window over into the View window.

Then drag a Label over.

In the View window, you can play around with the button and label just as you would in a graphics editing program. Go head and move them, resize them, and add some text to them. Double-clicking each object towards its center should allow you to add text.

When you're done, you should have something that looks like the image to the left.

Next, click on the Run button. Xcode will ask if we want to stop our current project. Go ahead and stop it, and let's see what our app looks like in the simulator!

A Visible Interface

Success! Our app has a visible interface now.

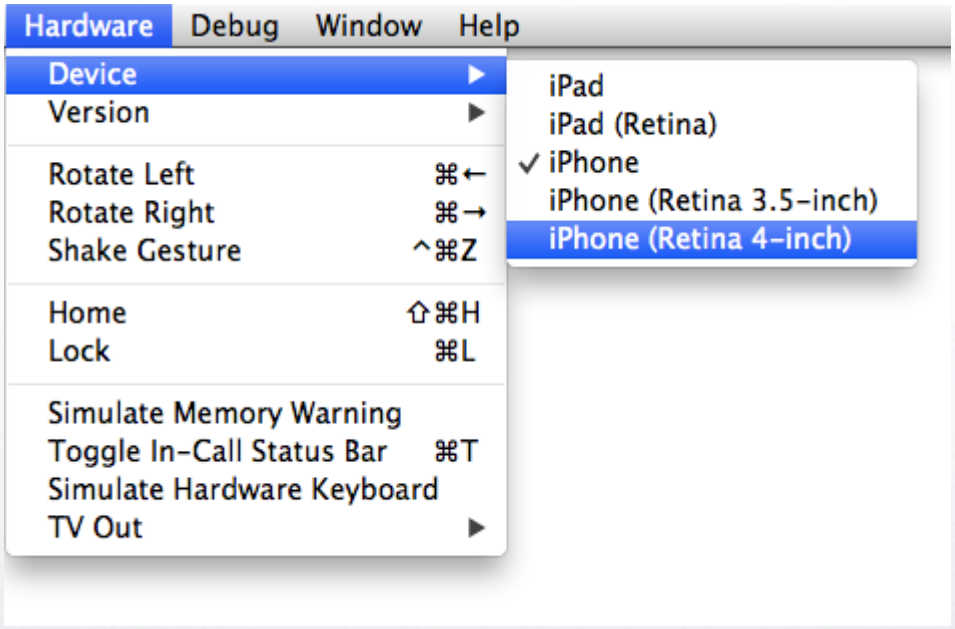
But wait a second... Does this look like what we created in Interface Builder? Or do the label and button overlap, which is different from what we saw in Interface Builder?

If the layout seems to have changed between Xcode and the iOS Simulator, the reason is probably that the simulator is showing the display for the original iPhone screen, but the default iPhone screen size in Xcode was for an iPhone 5.

Our app still runs, but iOS needed to move our controls in order to work on this kind of device.

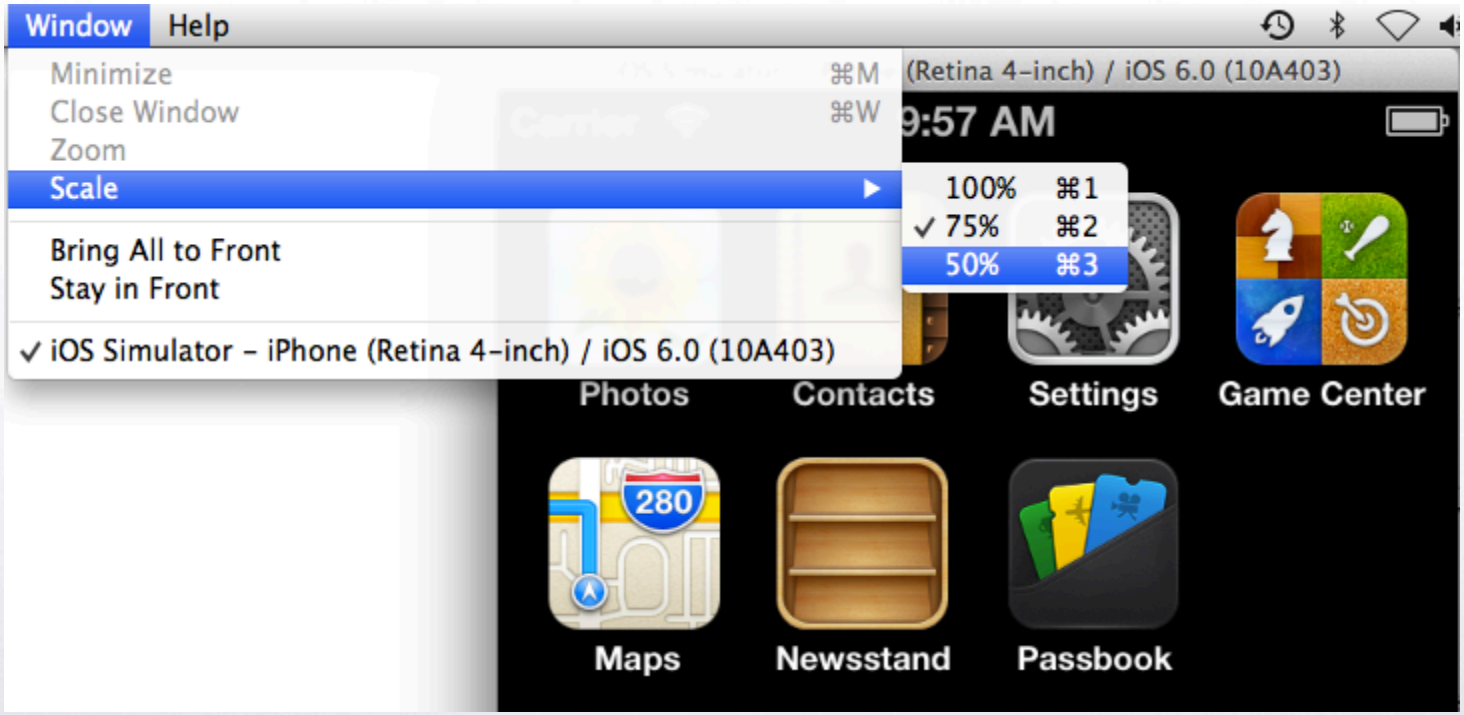
Simulating a different Device

In order to see how our app will look on an iPhone 5, let's choose the iPhone (Retina 4-inch) display from the Simulator's Hardware menu.



Too big?

If you're using a machine with a small screen, it may be that the iPhone Retina 4-inch display is too big to fit on your monitor. If this is the case, you can shrink it down by choosing a different Scale from the Simulator's Window menu.



Relaunching your App

When we changed the type of hardware that we were emulating, the Simulator automatically terminated our app and restarted the version of iOS that runs on the Simulator. To find and relaunch app, first swipe right to left in the Simulator to move to the second screen of icons. Once there, tap on your app's icon to launch it.

A Different View

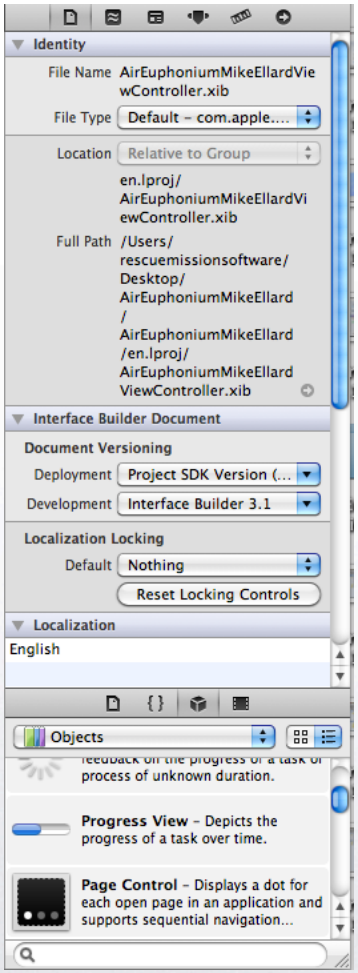
Success! Our app has a visible interface, and now it looks like we want it to.

Tap on the button in the iOS Simulator. The button highlights when you tap it, but it doesn't do anything more.

One of Objective-C's characteristics is that objects can be loosely linked to one another. The button we created doesn't care that we haven't linked it up to anything. It still knows that it is a button and that it should highlight when it is pressed.

Near the end of this project, we'll go back and check our layout to make sure that it will work correctly on both an iPhone 5 and an iPhone 4 or earlier.

The Inspector

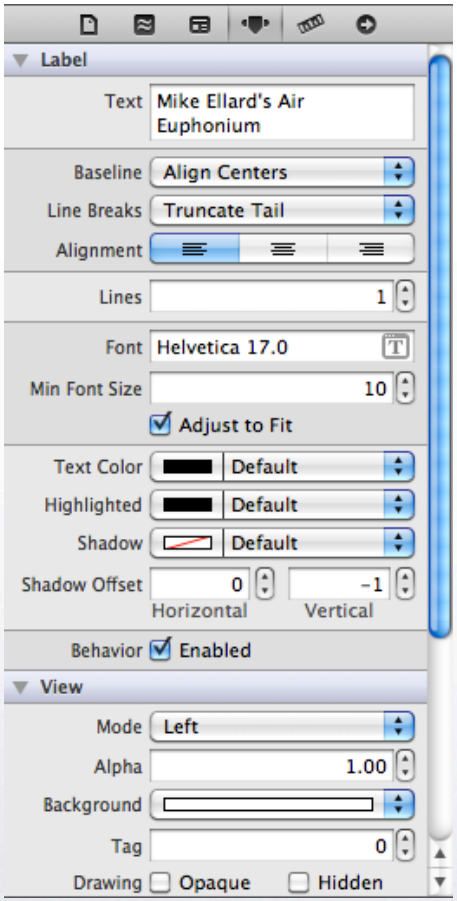


Next let's look at another Interface Builder component: the Inspector Pane. For this part of the lab, you may want to drag the boundary between the Library Pane and the Inspector Pane downwards to show more of the Inspector Pane.

The Inspector Pane has options to show an object's:

- File Information
- Quick Help. Links to the Apple documentation for this type of object.
- Attributes. Settings specific to the type of object we're looking at.
- Connections. This allows us to link our objects to the rest of our project so that our Interface Builder objects can interact with our Objective-C code and vice versa.
- Size. The object's size and position relative to the screen borders.
- Identity. The Objective-C class of the object and some other Interface Builder settings.

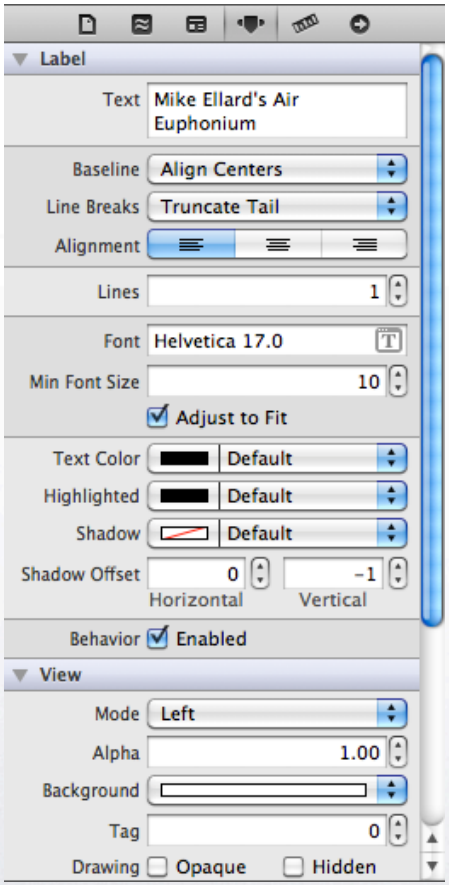
Exploring the Inspectors



Take a moment to explore the different inspectors by clicking the icons at the top of the Inspector Pane.

Try clicking on the button and label that we've added to our view to see how the inspector settings change between these two different object types.

Inspector or Graphical Editing?



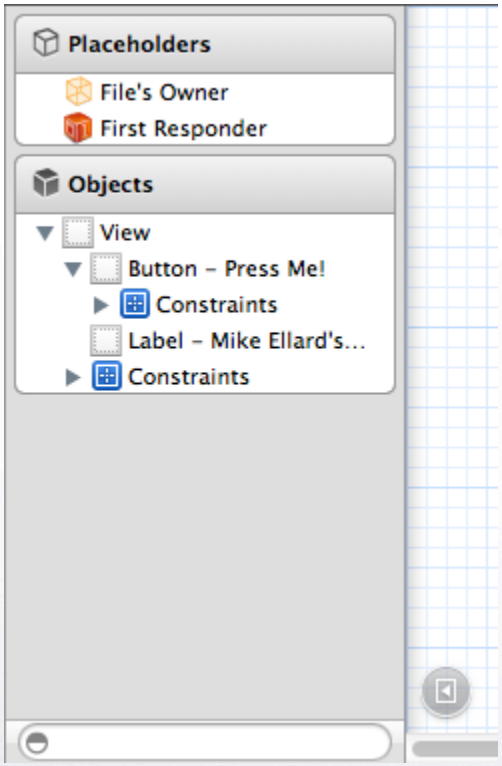
When we first dragged our button and label onto our view, we resized them and changed their text by dragging, clicking and typing just as we would in a graphics editor.

As you explore the Inspector, you'll note that you can also resize objects using the Size Inspector and change their text properties using the Attribute Inspector.

How should you edit objects: by clicking and dragging as you would in a graphics editor, or by typing in values in the appropriate Inspector fields? Probably you'll use a combination of both methods of editing objects when prototyping new apps or views.

Note: The Inspector allows you to change some attributes that you can't change via the graphical editor, so you should learn how to use the Inspector even if you prefer to use the graphical editor whenever possible.

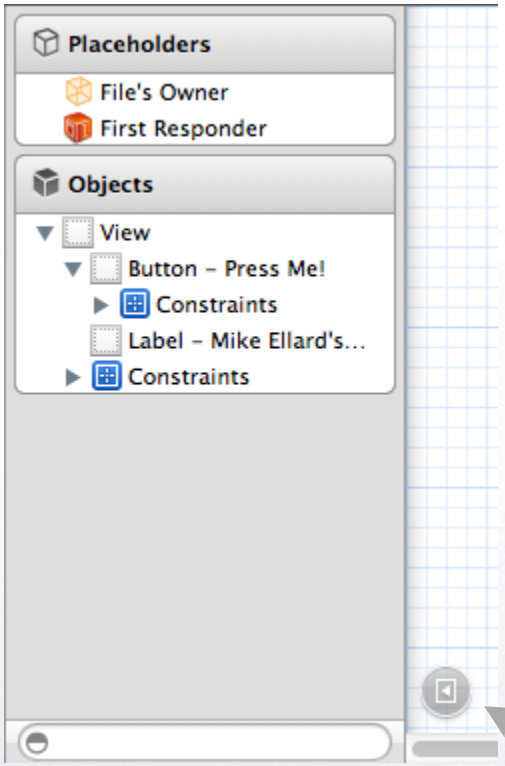
The Document Pane



At the right side of Xcode's main window pain is the Document Pane. It shows us three main objects.

- File's Owner. This is object represents the Objective-C class that is going to control the collection of objects we're creating in Interface Builder. This object is useful when you want to establish that interacting with a control is going to send a message to the controlling object (a.k.a. File's Owner) of the collection of objects that you're building.
- First Responder. This is a proxy object that represents the first responder in the responder chain that is used for event handling.
- View. This icon represents the view that is the container object for the collection of objects that we're working with in Interface Builder.

Views and Constraints



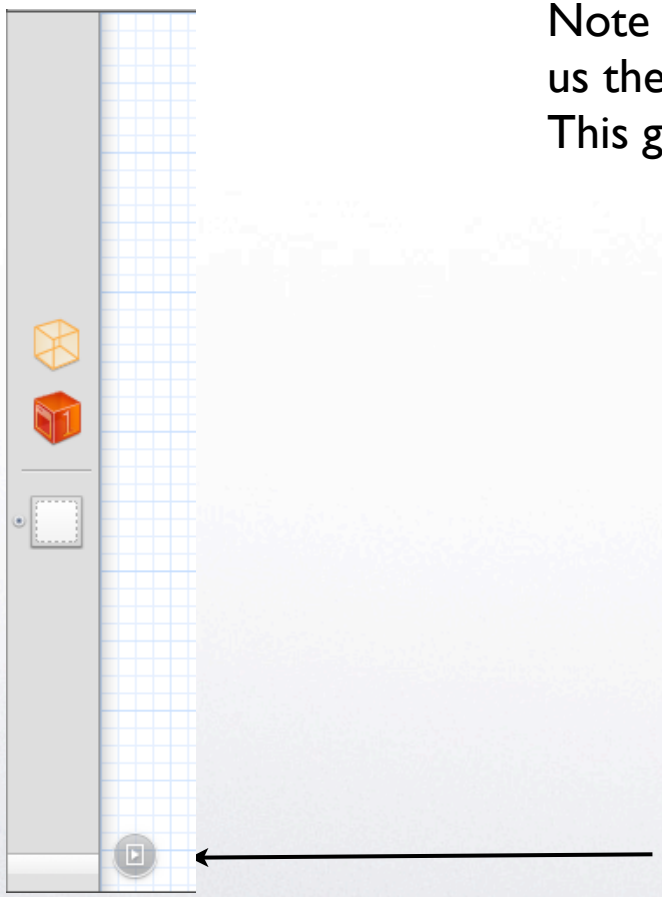
Since we added a button and a label to our UIView, we can see them in the view hierarchy below our the main view for the screen.

Note: You may need to click on the disclosure triangle next to the main view in order to see the full view hierarchy.

You also see some Constraints in the Document pane as well. Constraints are ways of managing the layout for objects under iOS 6. Unfortunately, Constraints don't work under earlier versions of iOS.

If the Document pane is taking up too much space in your screen, you can click this button to collapse it.

The Document Pane



Note that even when the Document Pane is collapsed, it still shows us the icons for the File's Owner, First Responder, and main UIView. This gives us access to these items even when the view is collapsed.

Click this button to make the Document Pane larger.

Let's hook up our button

Click on the ViewController.h file and add a method declaration so the .h file ends up looking like this:

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

@interface ViewController : UIViewController

-(IBAction)makeNoise:(id)sender;

@end
```

We've highlighted the new code in these examples in yellow so you can see what's changed. Your text won't be highlighted when you enter it.

Next, click on the ViewController.m file and add a method near the top of the file so it looks like this:

```
#import "ViewController.h"

@implementation ViewController

-(IBAction)makeNoise:(id)sender
{
    NSLog(@"I don't know how to do anything other than write to the console!");
}
```

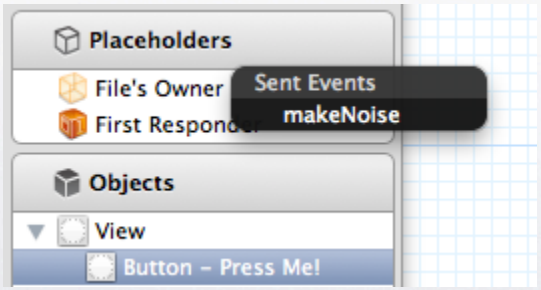
Connecting from Interface Builder

Choose the Build command from Xcode's Product menu.

Next, click on the ViewController.xib file.

Hold down the control key on your keyboard, then click down and hold down the mouse on the "Press Me!" button we created in the View window. Drag your cursor to the File's Owner icon in the Document Pane.

A blue line should follow us as we drag. Once we connect this blue line to the File's Owner icon, release the mouse button. You should see a small contextual menu that will allow us to connect the button with the "makeNoise" method that we want to call. Choose the "makeNoise" method, then choose "Save" from the "File" menu.

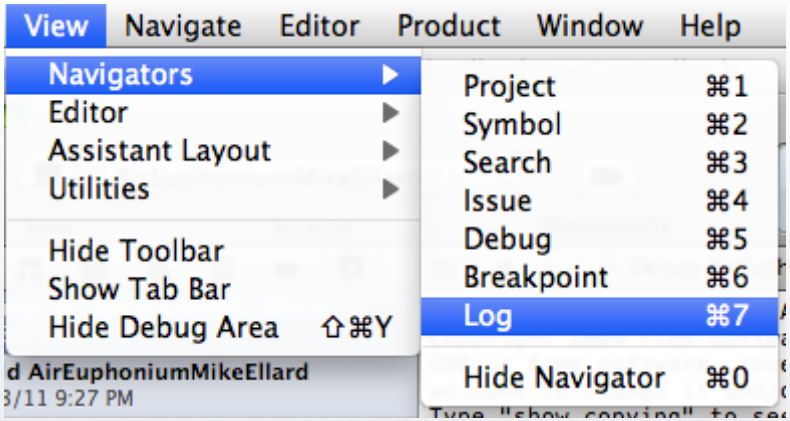


Viewing our Logs

Return to Xcode.

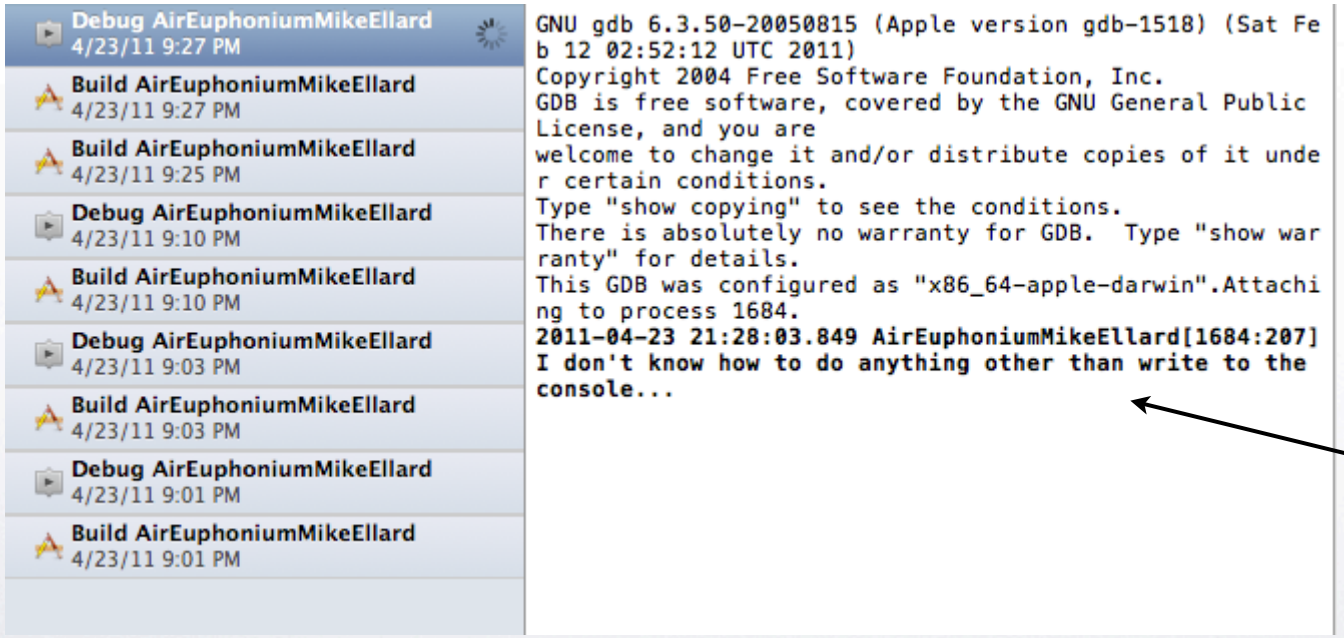
Hit the "Run" button and our app with its newly connected button should appear in the simulator. Tap on the button.

Now choose "Log" from the "Navigators" submenu of the View menu.



The Log Navigator

In the left pane of the navigator that appears, choose the top item. At the right side of the screen, we should see a console log that shows our message.



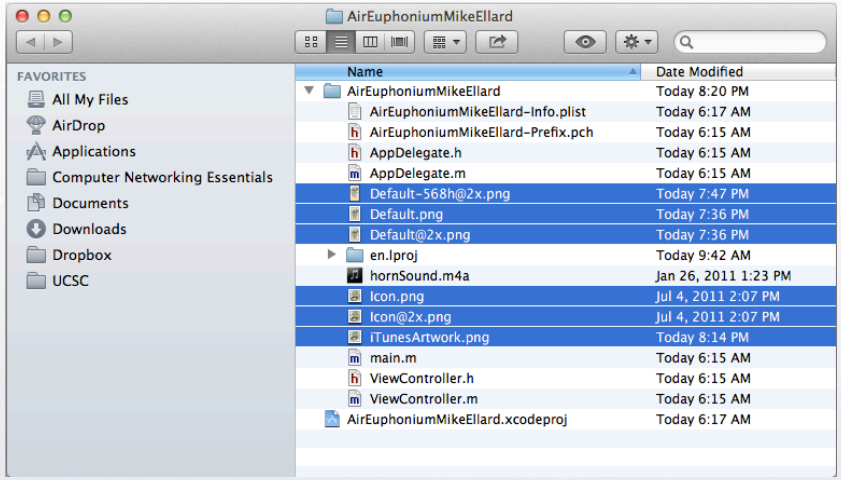
Copying files into our Project Folder

Next we're going to add some files to our project.

In the Resources section of our class website, you will find a zip archive called "Air Euphonium Assets." Download this archive and expand it. You should find seven files.

Now open the AirEuphonium Xcode project folder. It should have a second folder inside it with the same name. Copy the files from "Air Euphonium Assets" into this inner folder.

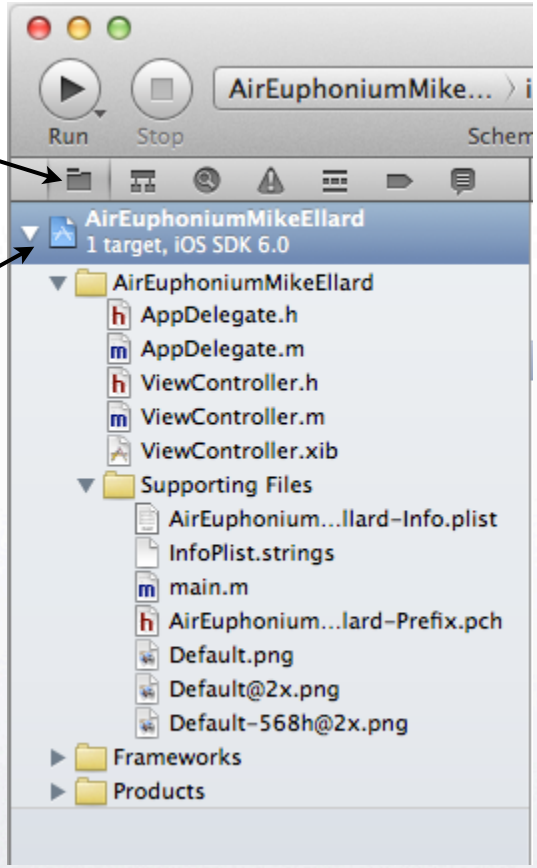
You should get an alert asking if you want to replace your existing Default.png files. Go ahead and replace them.



Adding Assets to our Project

Click here to show the Project Navigator if it's not already visible

Click here to select the project.



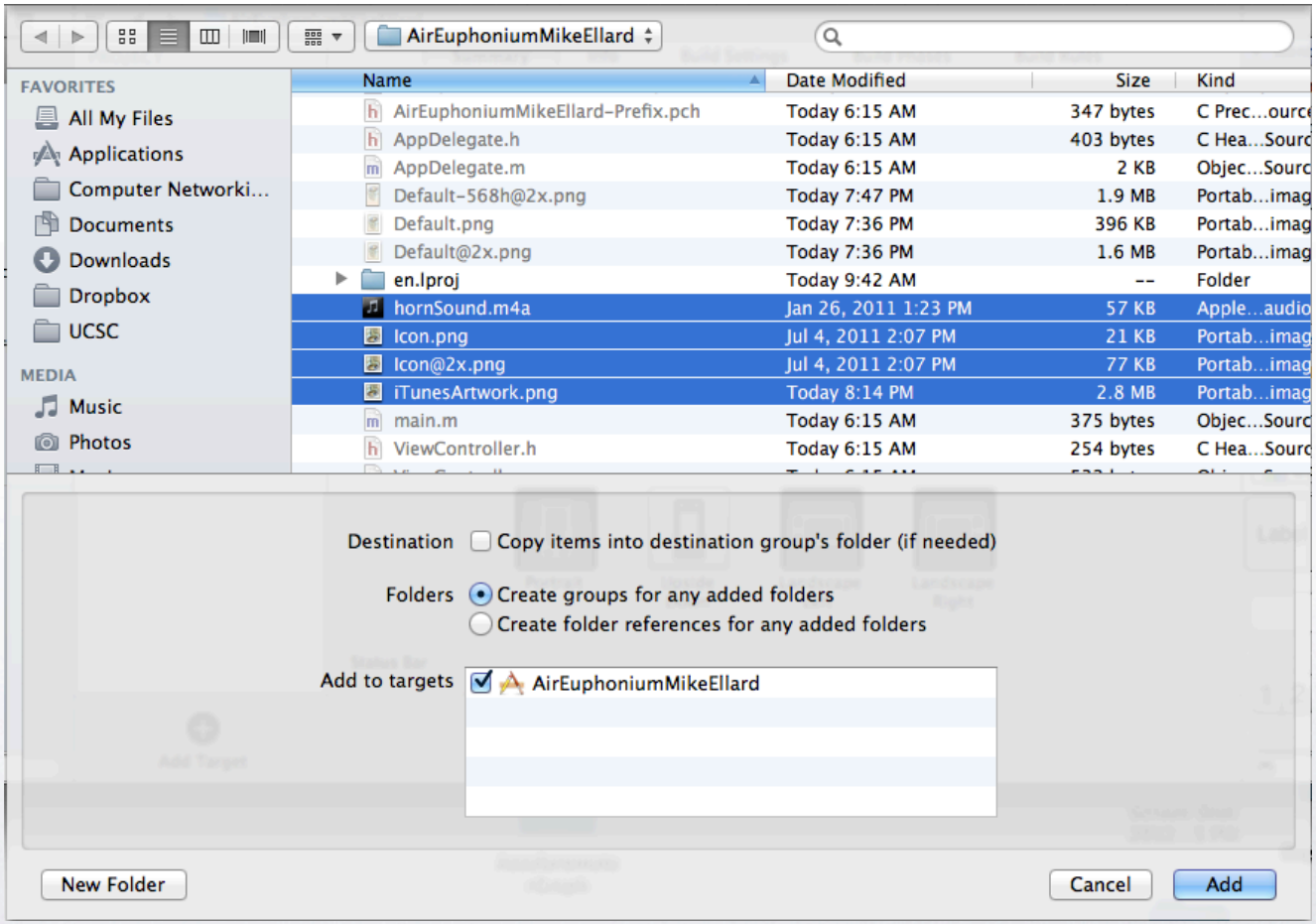
Next from within Xcode, select your project and then choose the "Add Files to AirEuphonium[YourNameHere]..." command from the File menu.

Although we copied the files into our project folder in the Finder, Xcode won't consider them to be a part of our project unless we explicitly add them.

Adding Assets to our Project

Finally, click add to add the icon, hornSound, and iTunesArtwork files to your project. They will now automatically be included whenever we build our project, whether or not we actually use them.

You don't need to explicitly add the Default.png files, since they replaced some files that were already part of the project.



Default.png

One of the files that we added was Default.png.

This image will show a startup screen whenever our project launches. Without it, the app would show a black screen at initial launch until our program had finished loading.

We don't need to do anything special to tell iOS about this file. iOS knows to expect a file called Default.png and to use it as a startup screen.

In order to be accepted in the Apple App Store, a program must include a Default.png.

@2x graphics & 568h@2x graphics

You'll notice that we have two versions of some of our graphics files:

- Default.png and Icon.png
- Default@2x.png and Icon@2x.png

The "@2x" in a graphic's file name indicates that it is high-resolution graphic for use on the high resolution screens of the iPhone 4 and 4th Generation iPod Touch. In most cases, if you include both a regular and an @2x graphic in an application, iOS will automatically use the appropriate graphic for the device on which your app is running.

The Default-568h@2x graphic is for use on the iPhone 5.

Adding Icons to our Info.plist

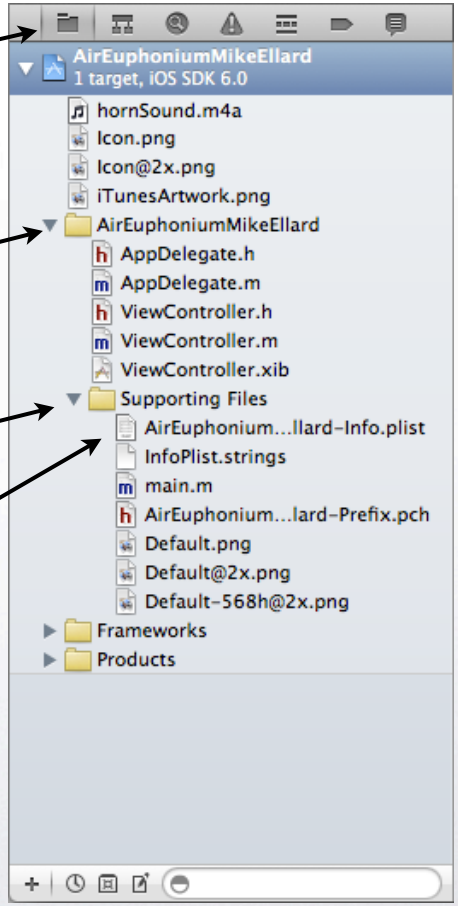
Unlike Default.png, we need to explicitly include our icons in our Info.plist file if we want them to be picked up in our app.

Click here to show the Project Navigator if it's not already visible

Then click here to open the AirEuphonium[YourNameHere] File Group

Then click here to open the Supporting Files file group

Then click on the AirEuphonium[YourNameHere]-Info.plist file

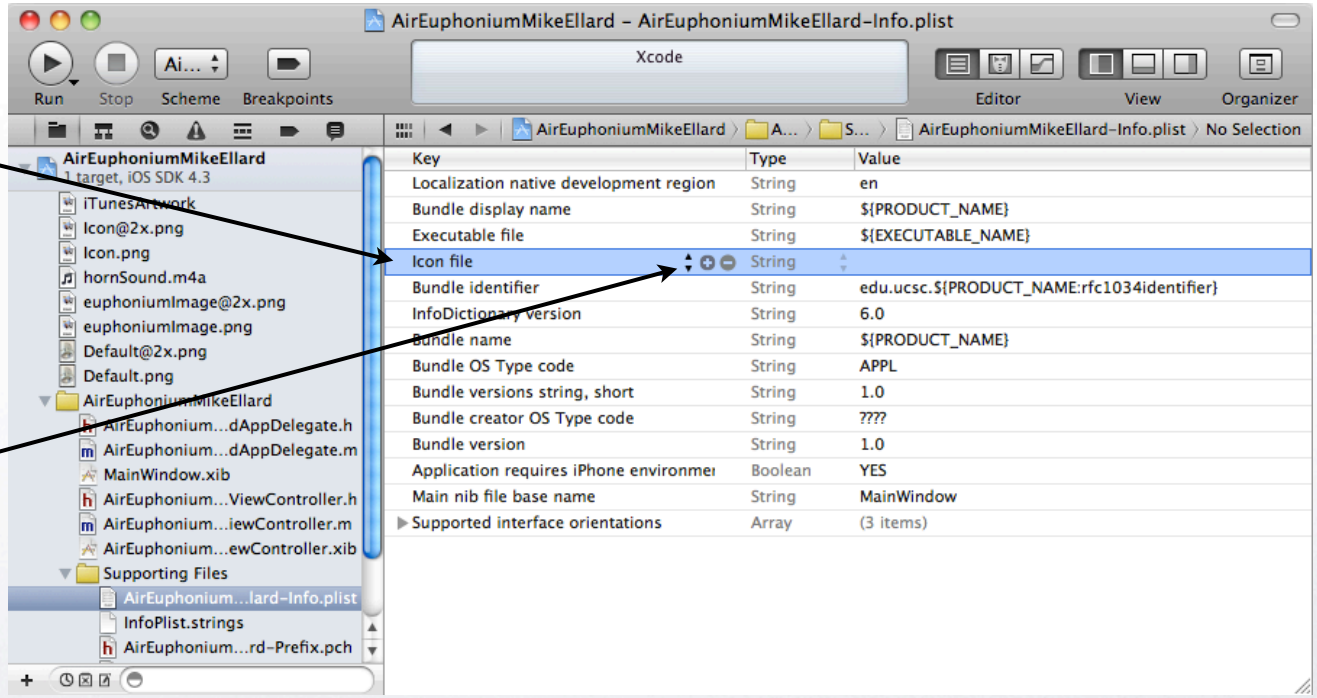


Adding an item to our Info.plist file

You'll see an item on our Info.plist called Icon file, but our app is going to contain more than one icon, so we're going to add an array item called Icon files.

First click the Icon file line

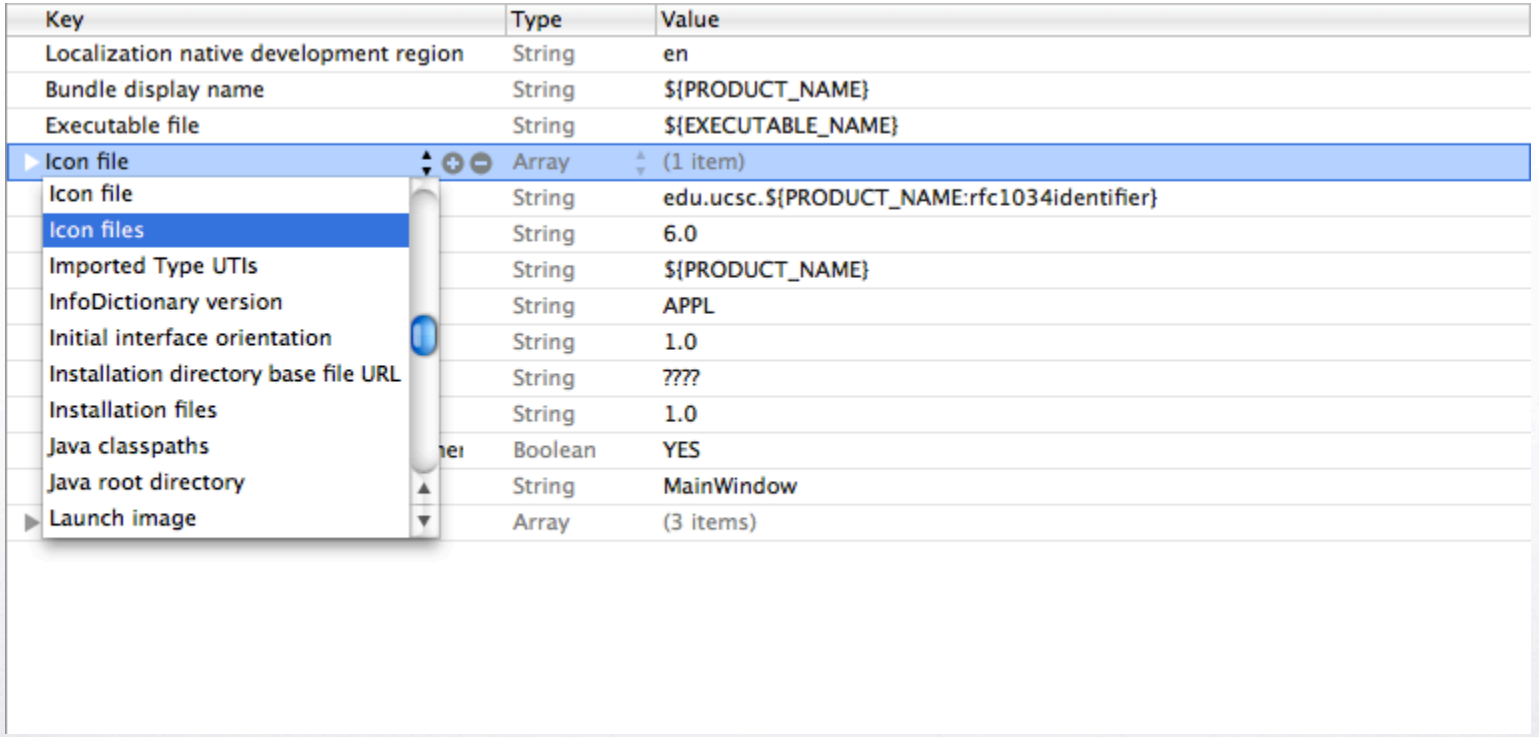
Then click the up and down arrows to allow us to change this item's type.



Note: If there is not an Icon file item, click in any row and then click the "+" sign to add a new row. In the new row, select Icon files as the type.

Choosing a type for our new item

Choose the Icon files type in the menu that appears.

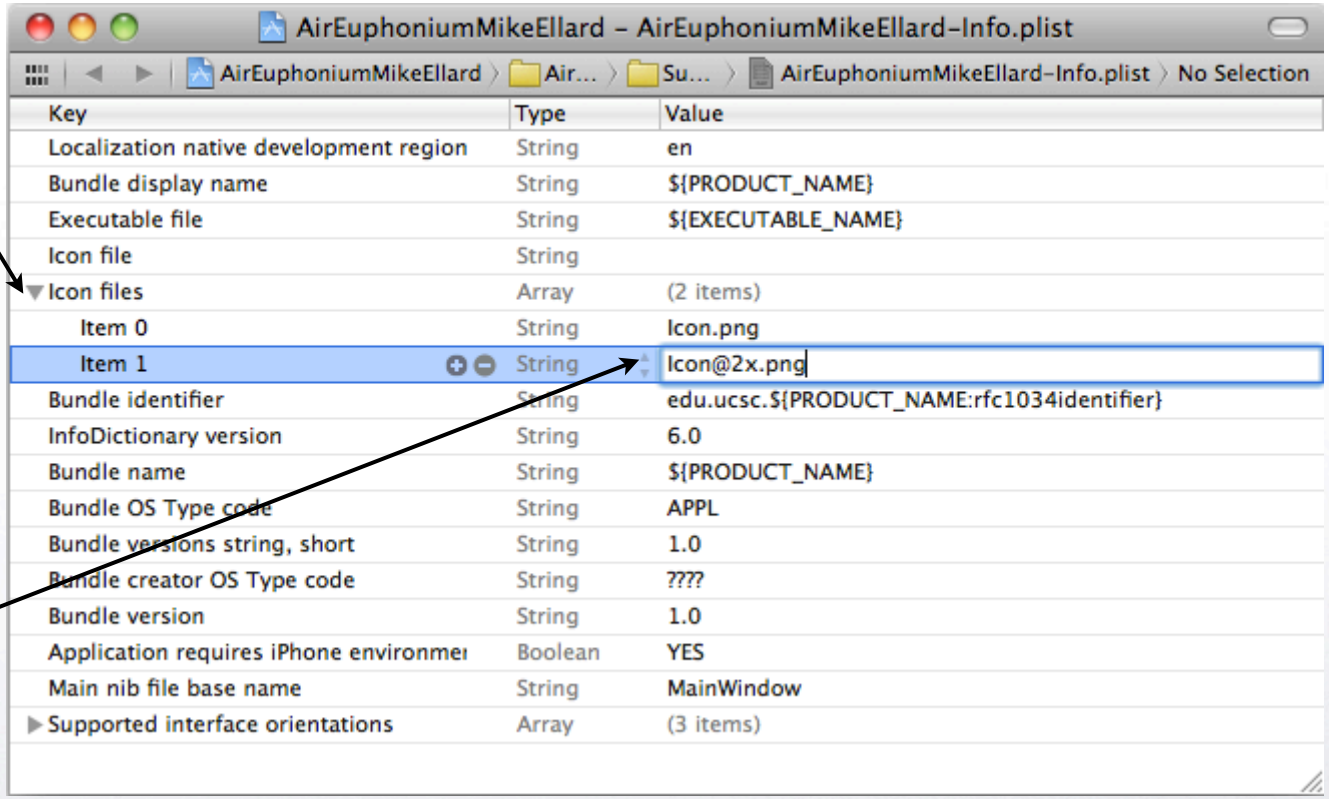


Entering our icon information

First click the detail disclosure triangle to see the Icon files array items.

Next click the plus button on Item 0 to add a new item.

Finally enter your file information. File names on iOS are case sensitive.

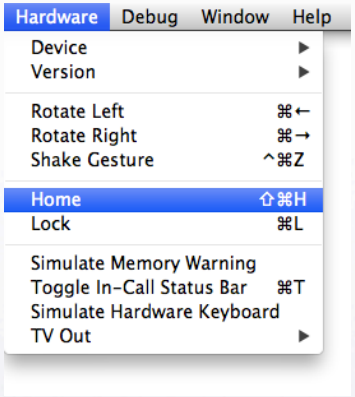


Testing your work

Now test your work. Build and run your app. It should now have a splash screen that shows up when you first run the app.

If you press the home button in the simulator, you should see that your app has a snazzy icon!

To simulate pressing the home button in the iPhone 5 Simulator, choose the Home command from the Simulator's Hardware menu.

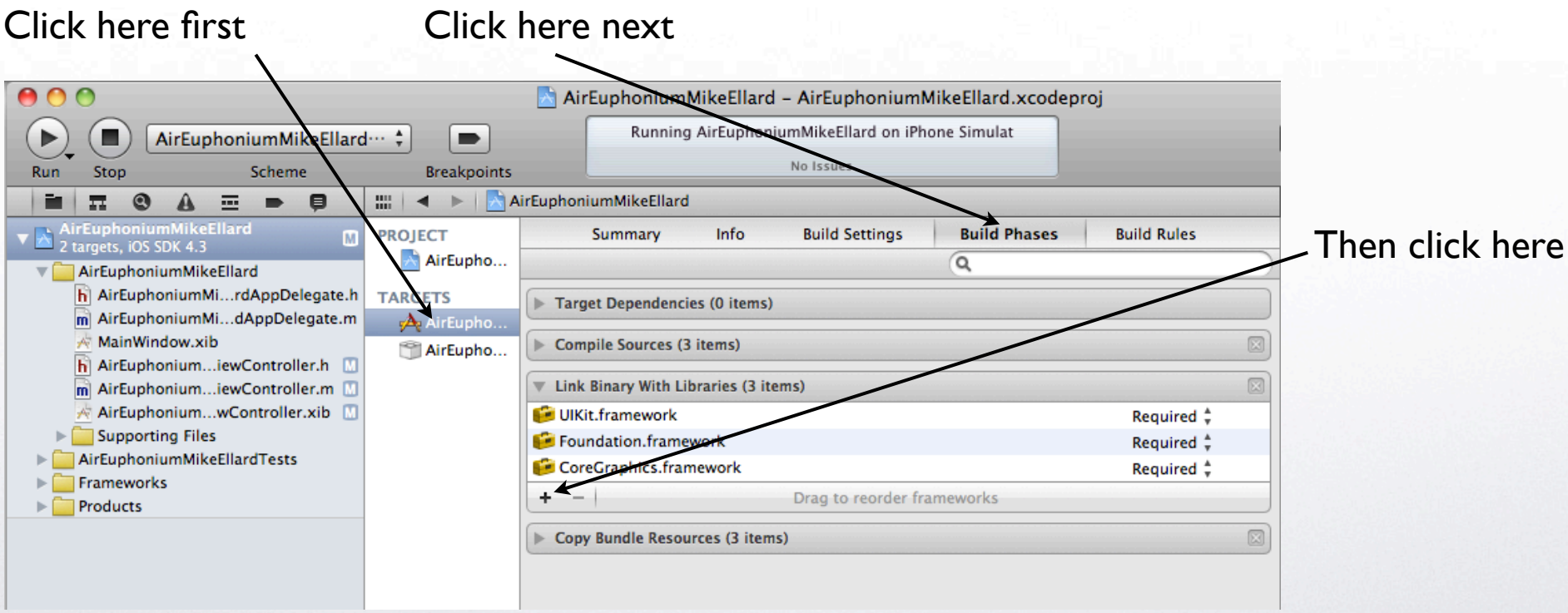


When you've got everything working, let's return to Xcode so that we can add some more functionality to your app.

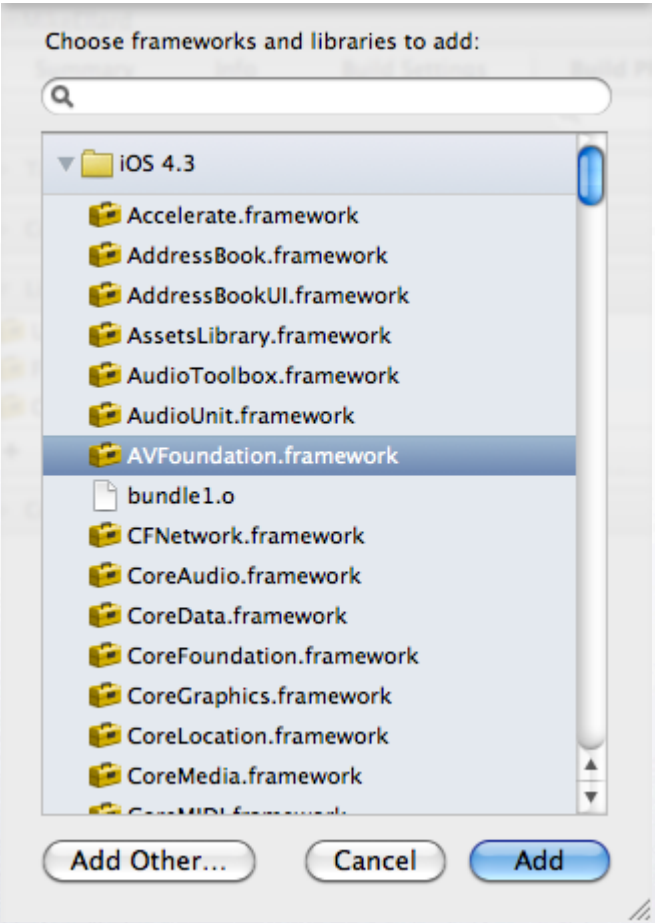
Adding a Framework to our Project

In order to make noise, we're going to add the AVFoundation Framework to our project.

To add a framework, follow the steps below:



Choosing our Framework



Finally, choose the AVFoundation.framework and click the Add button.

Updating our .h File

Now let's update our ViewController.h file so that it looks like this. If you're still looking at Frameworks, click the Classes folder at the right side of the project window to see the file you want. Click on the file, and make the follow edits:

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

@interface ViewController : UIViewController

-(IBAction)makeNoise:(id)sender;

@property (nonatomic, retain) AVAudioPlayer *hornSound;

@end
```

Updating our .m File

Now let's update our AirEuphonium[YourNameHere]ViewController.m file. First we're going to make some changes at the top of the file:

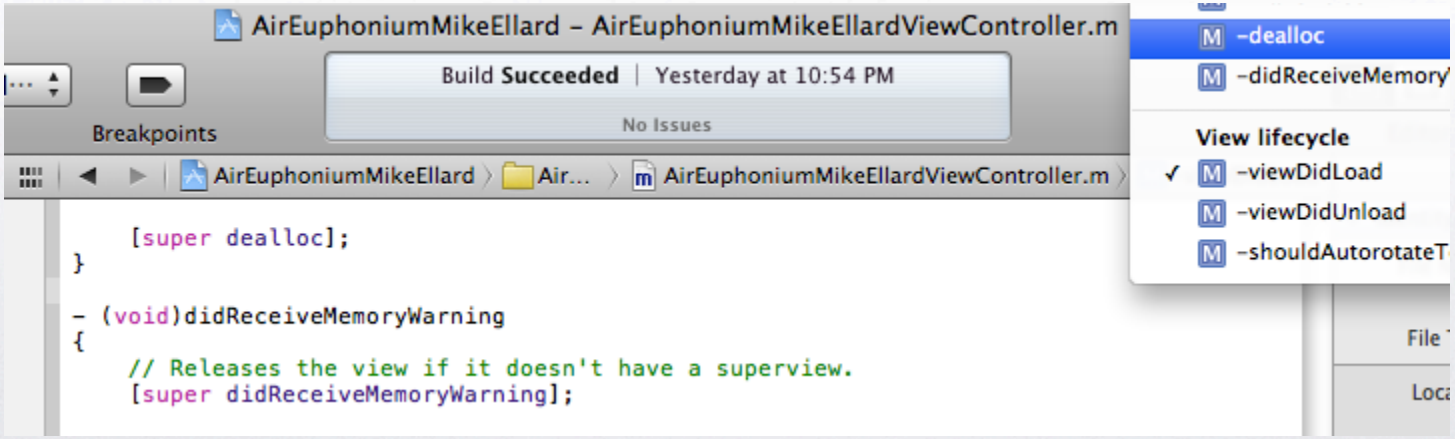
```
#import "ViewController.h"

@implementation ViewController

@synthesize hornSound;

-(IBAction)makeNoise:(id)sender
{
    [hornSound play];

    //NSLog(@"I don't know how to do anything other than write to the console!");
}
```

The Jump Bar

Next, we're going to need to change the -viewDidLoad method.

To get there, we can use the Jump Bar. At the top of your code pane, there is a Jump Bar that shows the method you are currently editing, the file that it belongs to, and that file's position in your project's Xcode hierarchy. By clicking on the method name, you will see a menu with a listing of all of the methods in your current file. You can can use this to easily jump to any of the other methods in your file.

Note: This is just the beginning of what you can do with the Jump Bar. You will use it a lot!



Changes to viewDidLoad

Next we're going to update the viewDidLoad method.

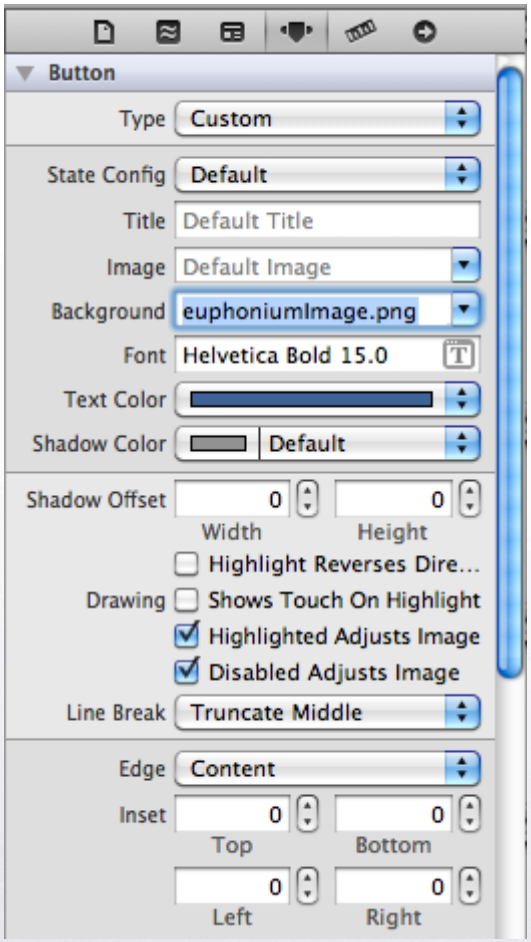
```
- (void)viewDidLoad
{
    [super viewDidLoad];

    NSError *err = nil;

    NSBundle *bundle = [NSBundle mainBundle];

    NSString *path = [bundle pathForResource:@"hornSound" ofType:@"m4a"];

    NSURL *url = [NSURL fileURLWithPath:path];

    hornSound = [[AVAudioPlayer alloc] initWithContentsOfURL:url error:&err];

    if (err)
    {
        NSLog(@"%@", err);
    }

    [hornSound prepareToPlay];
}
```

Creating a dealloc method

Finally, add a dealloc method:

```
- (void)dealloc
{
    [hornSound release];
    [super dealloc];
}
```

Polishing our Interface...



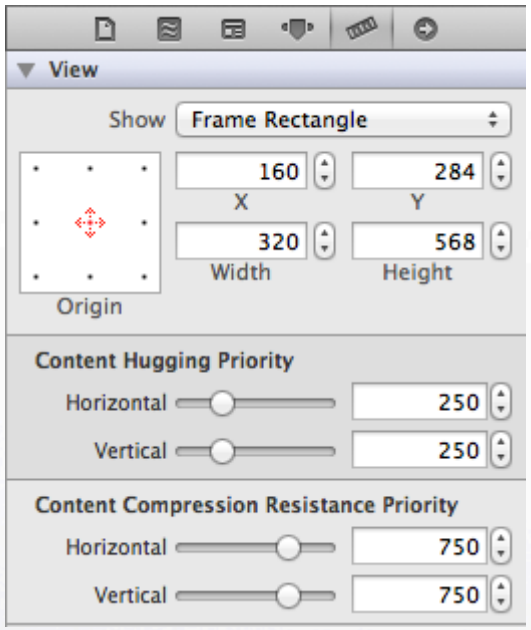
Run the project with your edits. Your button should make a sound when you tap it!

All that's left to do is make it look like a euphonium!

Back in Xcode, click on your XIB file to edit it.

Click our button, then choose the "Attributes Inspector" from Inspector Pane. Change the Type to "Custom," blank out the Title attribute, and choose Default-568h@2x.png from the Background menu.

Resizing Our Button



Next choose the "Size Inspector" from the Tools menu.

Enter 160 for X, 284 for Y, 320 for W, and 568 for H.

This should make our button fill the screen.

You should reposition your label text at this time. Put it near the top of the screen, then play with that until it looks right to you.

Save your changes in Interface Builder, go back to Xcode, Build and Run.

Make sure the positioning setting in the upper left of the pane is as shown here so that you're positioning the graphic relative to the center of the window.

Run and Check Your App

Congratulations - your app looks and sounds great!

But there's one more thing to do. We need to check it on the iPhone simulator to make sure it will look okay on earlier devices.

Use the Simulator's Hardware menu to switch to an "iPhone (Retina 3.5 inch)" device.

Find the AirEuphonium icon and relaunch your app.

Congratulations!

By completing these instructions you have:

- Worked with Xcode
- Written some Objective-C Code
- Added graphic and audio resources to a project
- Added an iOS framework to a project
- Built a project using Xcode
- Run an app in the iOS Simulator
- Created an app that could go to Number 1 on the App Store!

Excellent work!