How To Use Blocks in iOS 5 Tutorial – Part 1

Adam Burkepile on February 8, 2012

This is a blog post by iOS Tutorial Team member Adam Burkepile, a full-time Software Consultant and independent iOS developer. Check out his latest app Pocket No Agenda, or follow him on Twitter.

Blocks are an incredibly powerful extension to C/Objective-C. They allow you to wrap up chunks of code in self-contained units and pass them around as objects.

More and more APIs are requiring blocks in iOS, so you really need to understand them to do almost anything. However, the syntax and subtle aspects are often confusing to beginners. Never fear – that's where this tutorial comes in! :]

In this two-part tutorial series, you're going to create a little iOS project called the iOS Diner. The app itself will be simple: users choose items from a menu to create an order, as if they were about to eat at a diner. Be warned, this project might make you hungry!

In this first part the tutorial we'll create the app's UI. This will be a review of iOS 5 Storyboards, including a brief refresher of how to create and use web services to download the diner menu in JSON format.

Order up some Storyboards and Blocks in this tutorial!

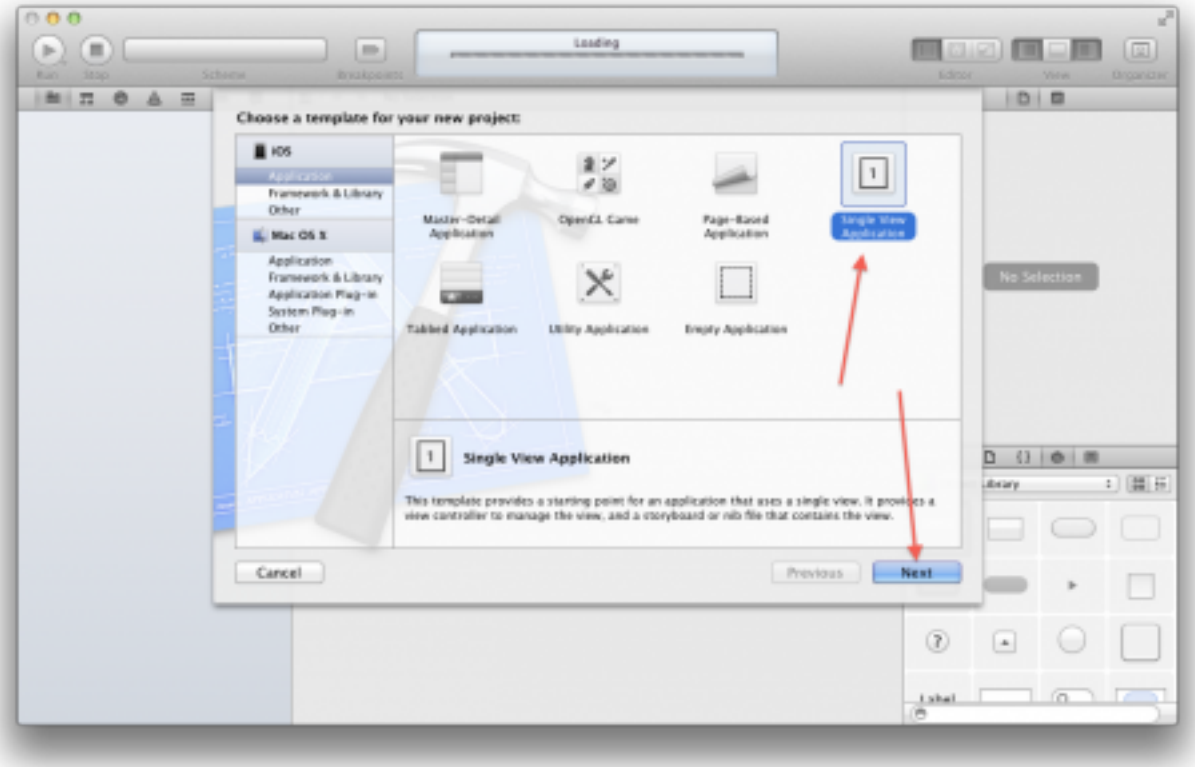
Note: If you are already pretty comfortable with Storyboards and Interface Builder, you may want to skip Part 1 and go straight to Part 2, where we start using blocks. This part focuses on Storyboards and Interface Builder only.

The second part of the tutorial will make extensive use of blocks to code the app's logic. It will demonstrate using blocks for asynchronous processing, background tasks, alternatives to more standard APIs, and much more.

If you're hungry to know more, keep reading!

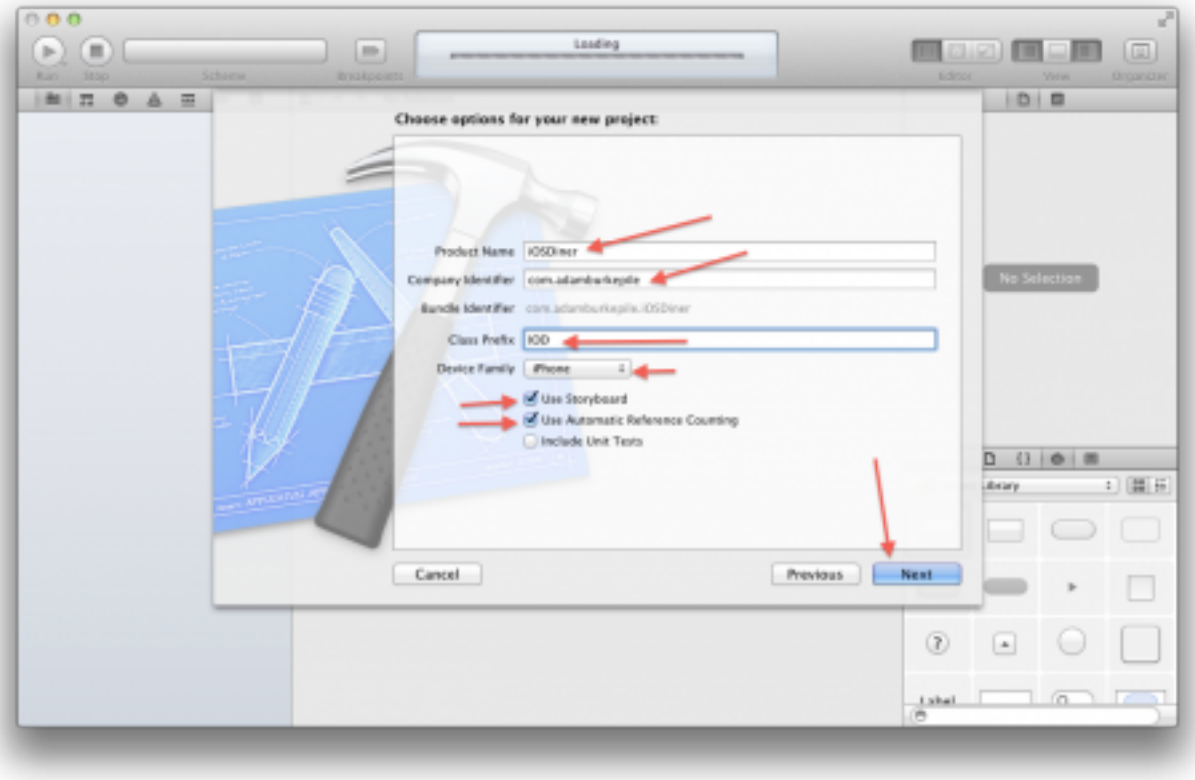
Getting Started

First, let's crack open Xcode and start a new project with the iOS\Application\Single View Application template.



Enter “iOSDiner” for the product name, enter the company identifier you used when creating your App ID, and enter “IOD” for the class prefix.

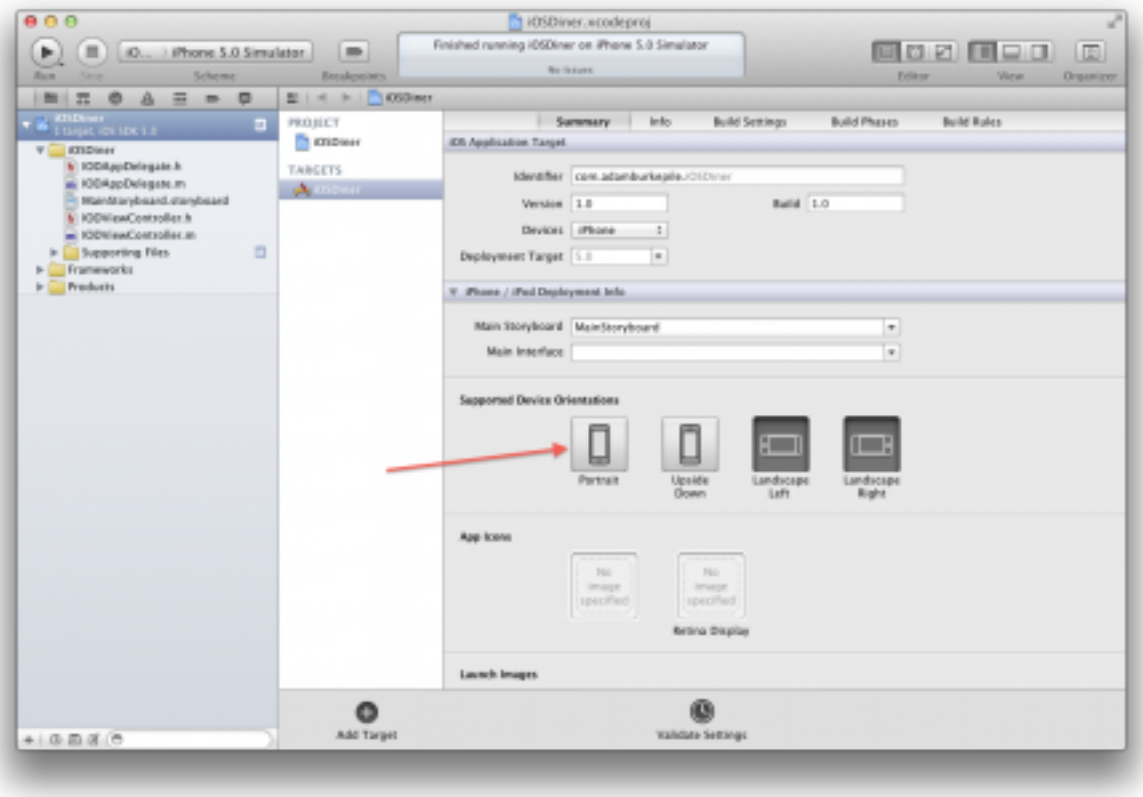
Set device family to iPhone only. Make sure Use Storyboard and Use Automatic Reference Counting are checked, but that Include Unit Tests and any other checkboxes are unchecked.



The next screen asks where you want to save the project. You can save it anywhere.

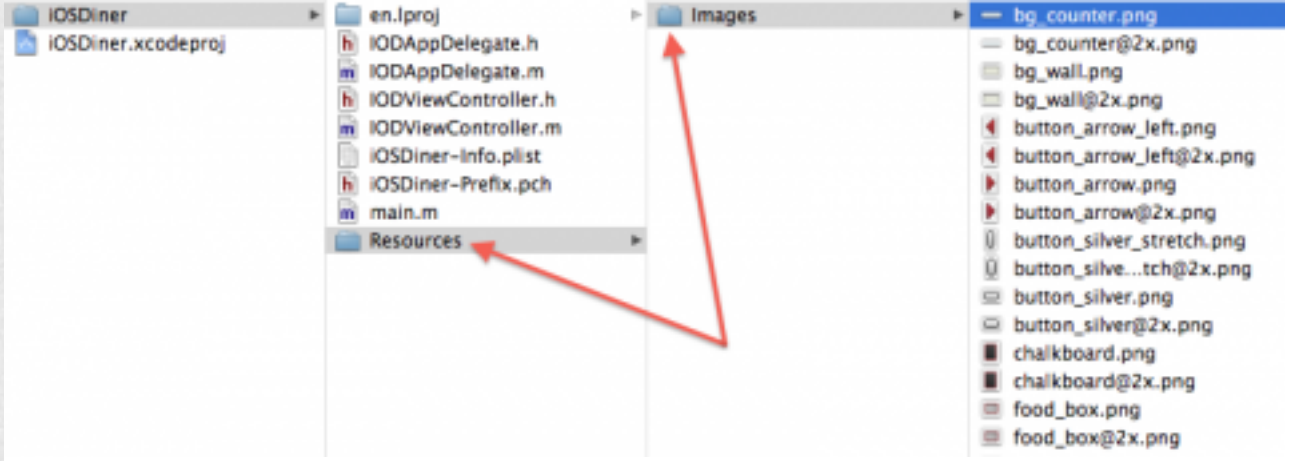
Give it a quick run and you should see a blank screen.

Under the Supported Device Orientations section of the iPhone/iPod Deployment Info menu (on the Summary tab), deselect Portrait mode so the app only runs in landscape.

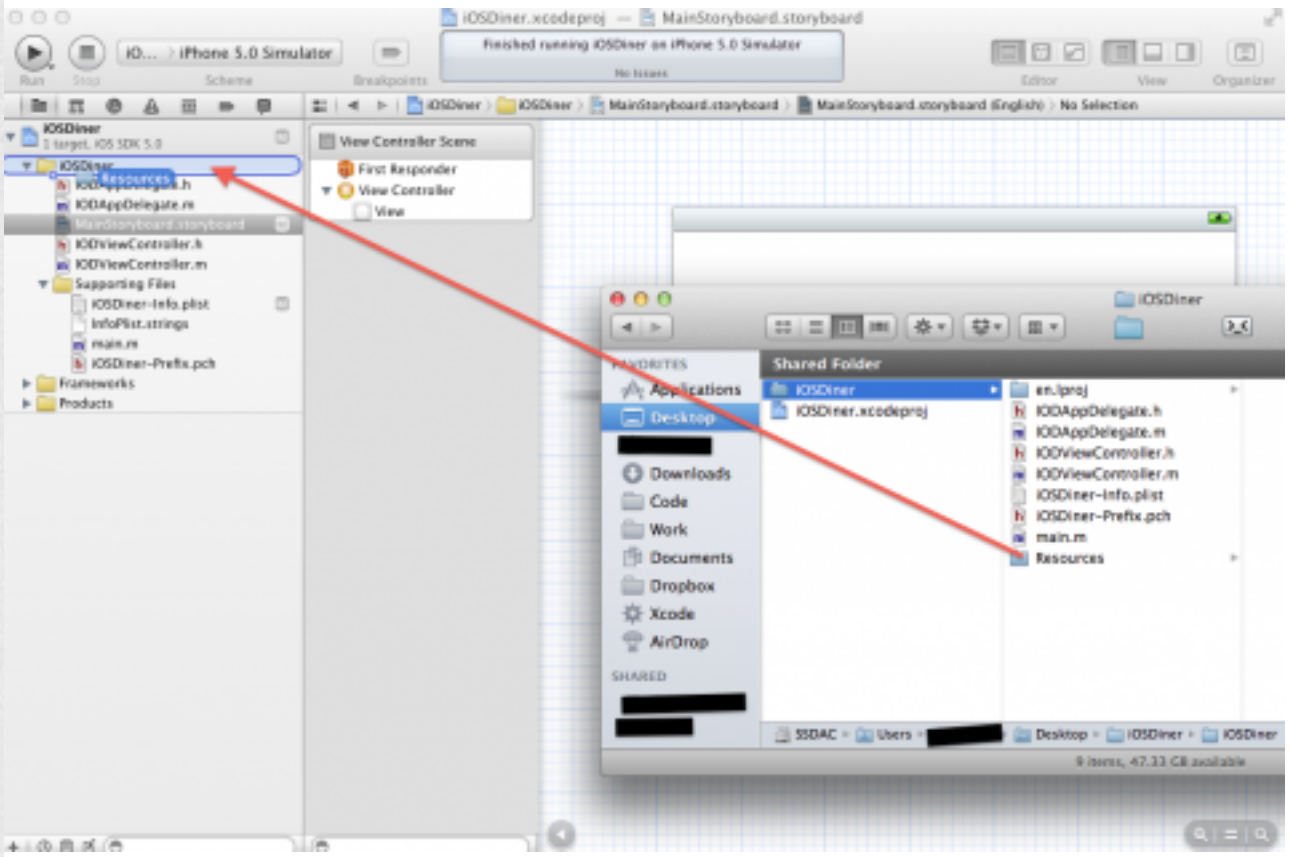


The first thing you'll do is set up the view. In order to do this, we need some graphics. We've been provided with some wonderful art by Ray's wife Vicki – you can download it here. We need to add the art resources to the project.

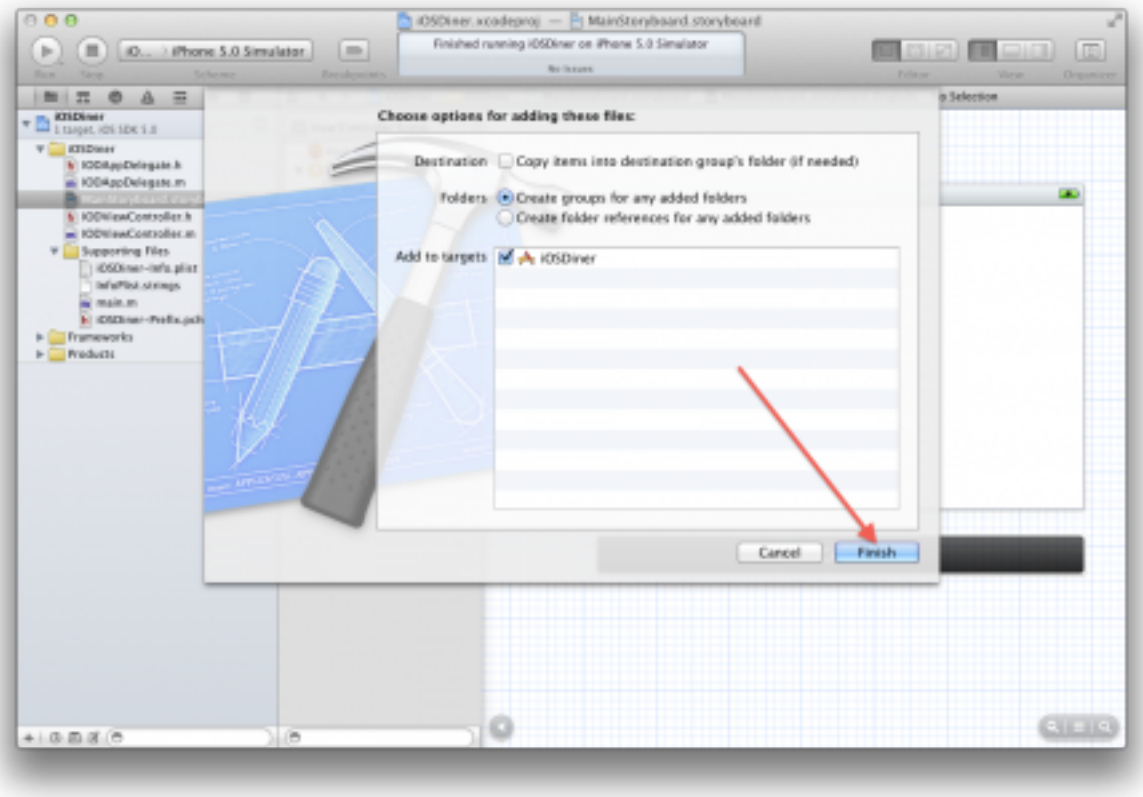
I've never really liked how Xcode handles matching files in the project and in the file system, so I add resources manually in the files system most of the time. To do this, use Finder to navigate to the project folder. Create a folder called "Resources" inside the project folder. Then create a folder called "Images" inside the Resources folder.



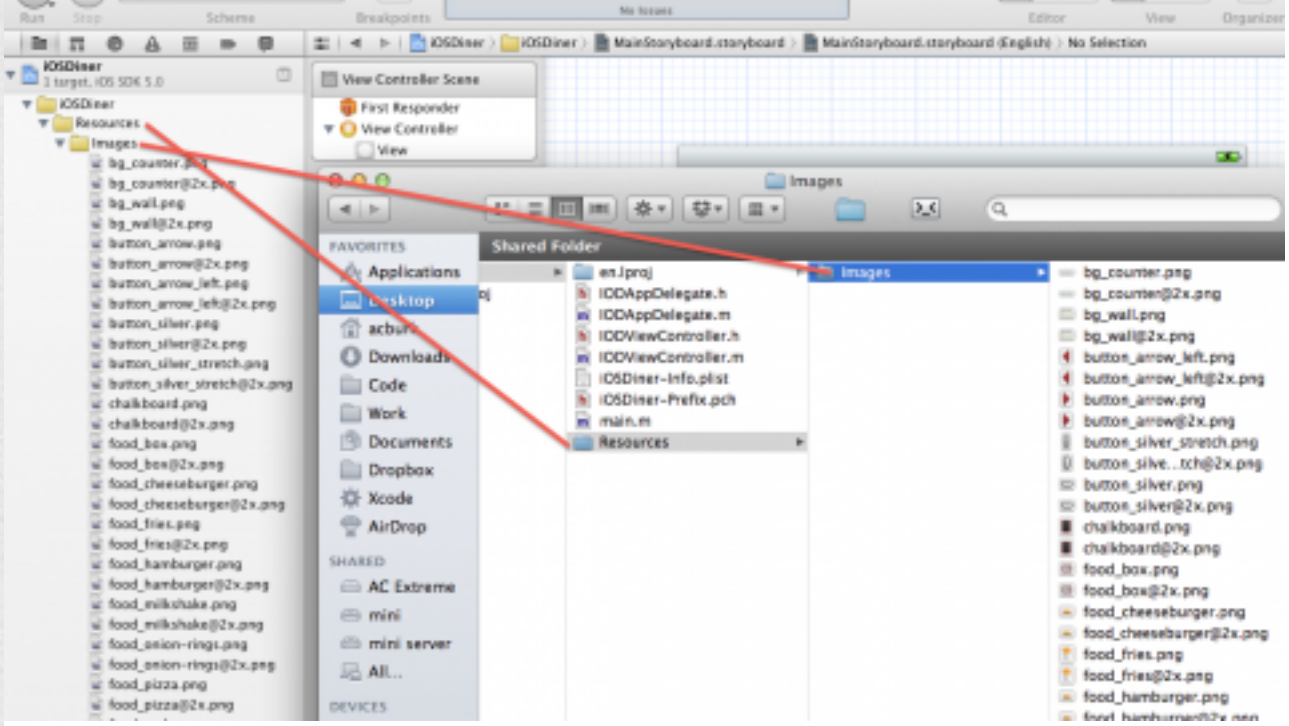
Copy the art from the ZIP file you downloaded into the Images folder, then drag the Resources folder into the iOSDiner folder in Xcode as shown in the screenshot below.



Make sure that the next dialog for adding files is configured as per the screenshot below and then click “Finish” to add the files.

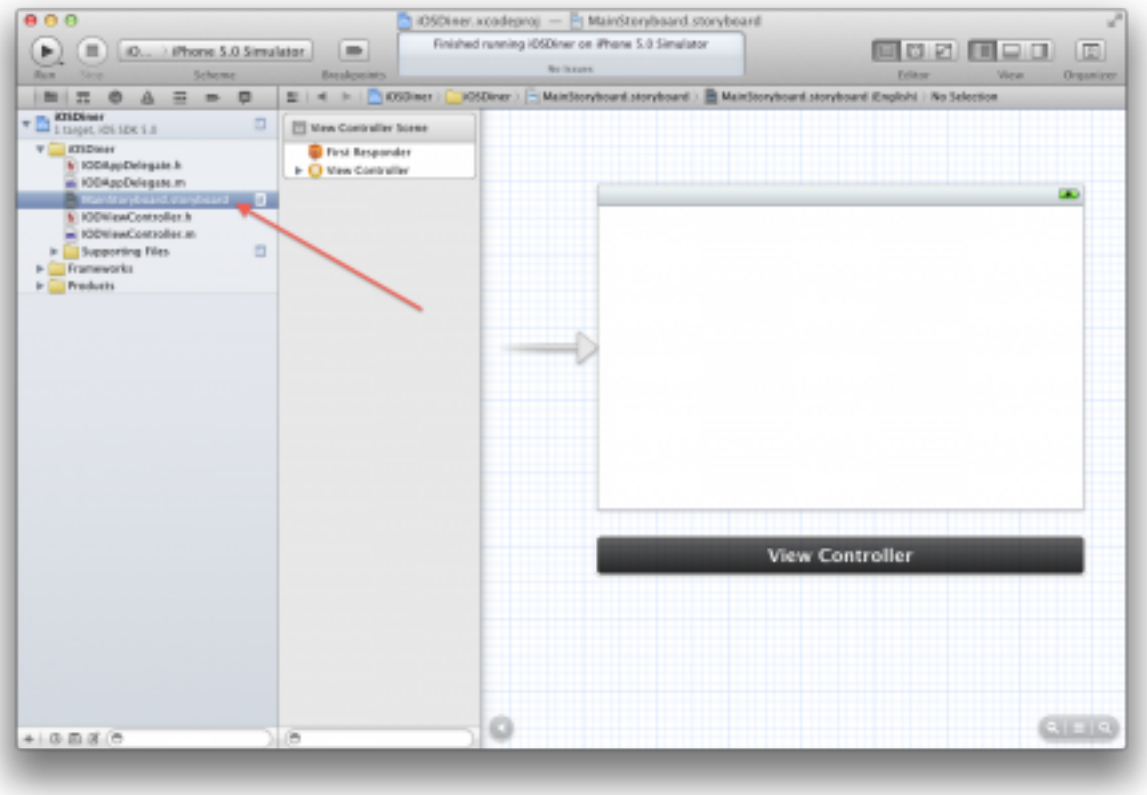


Now you should see a Resources folder inside Xcode that has an Images sub-folder inside it and which in turn contains the images you downloaded – exactly the same way the folders are structured under the file system.

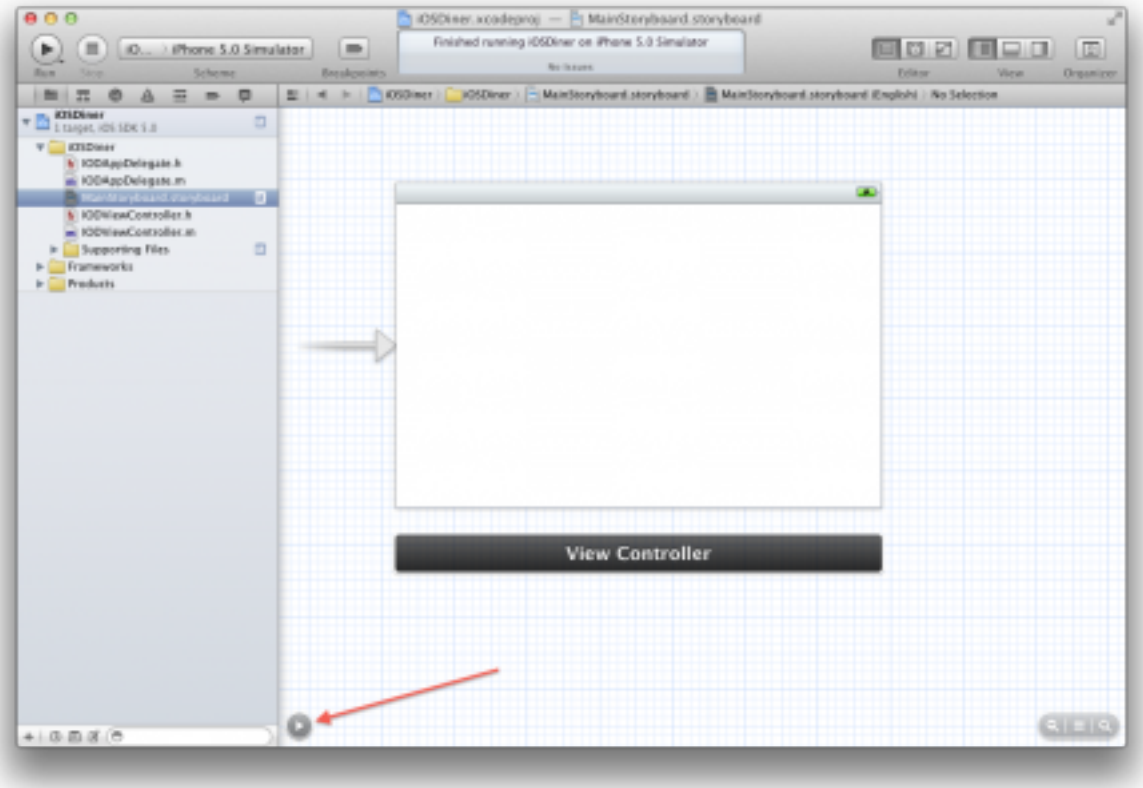


Adding Images

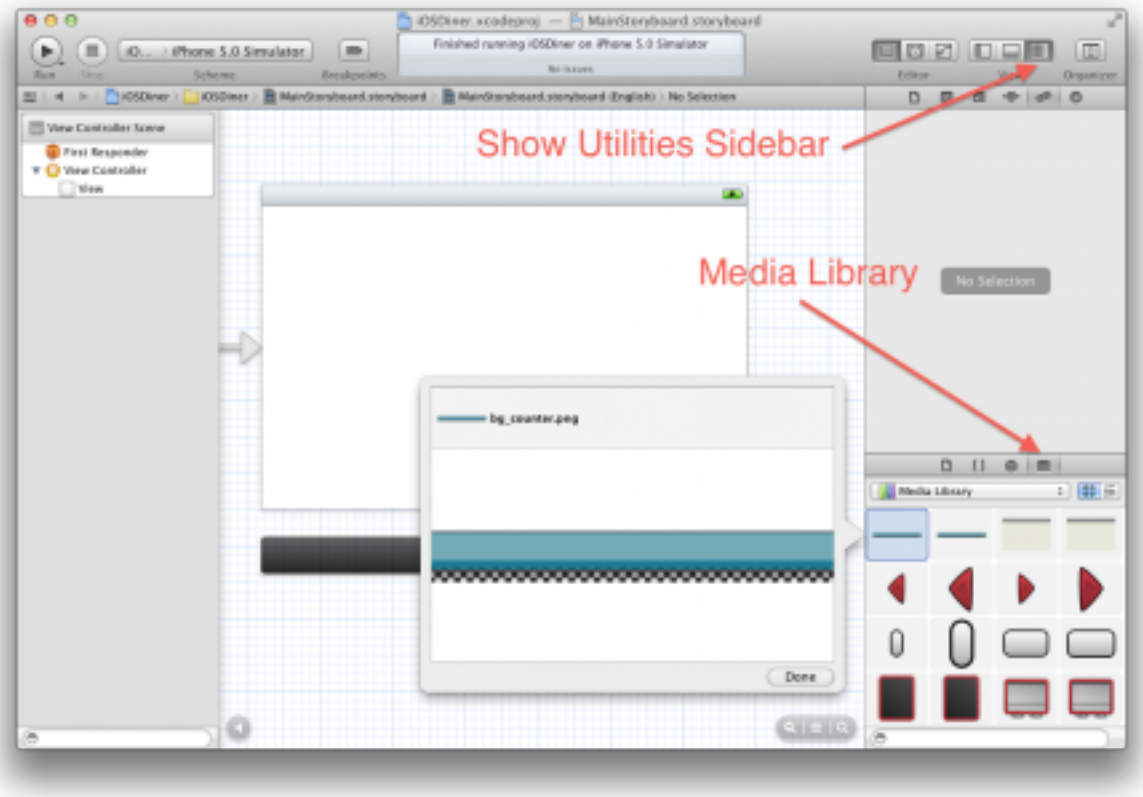
Select MainStoryboard.storyboard.



If you don’t see that second column that says “View Controller Scene,” click the Expand button at the bottom (as indicated in the image below).



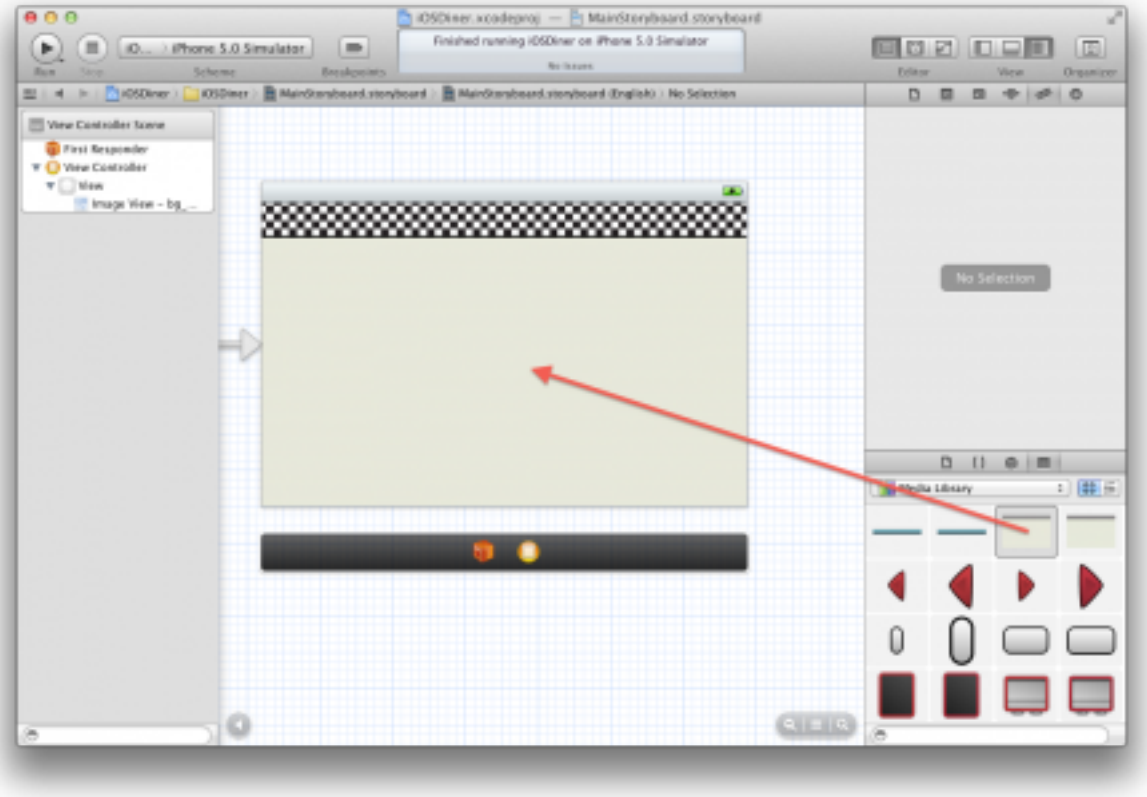
You are going to add the images to the storyboard in the form of UIImageViews and UIButtons. To make things easier, expand the Utilities sidebar and select the Media Library.



Here we see all the images that we added to the project earlier. You'll notice there are two of every image. This is because there is a normal and a retina version (@2x) of every image.

We're only concerned with the normal versions. You can check which is which by selecting an image and pressing the space bar. You'll see a pop-up window. The images **without** @2x in the name are the normal versions.

Drag "bg_wall.png" onto the root view and place it as indicated below. If you aren't sure that the image is placed correctly, you can switch to the Size Inspector (the fifth tab in the Utilities sidebar) to specify exact X and Y coordinates.

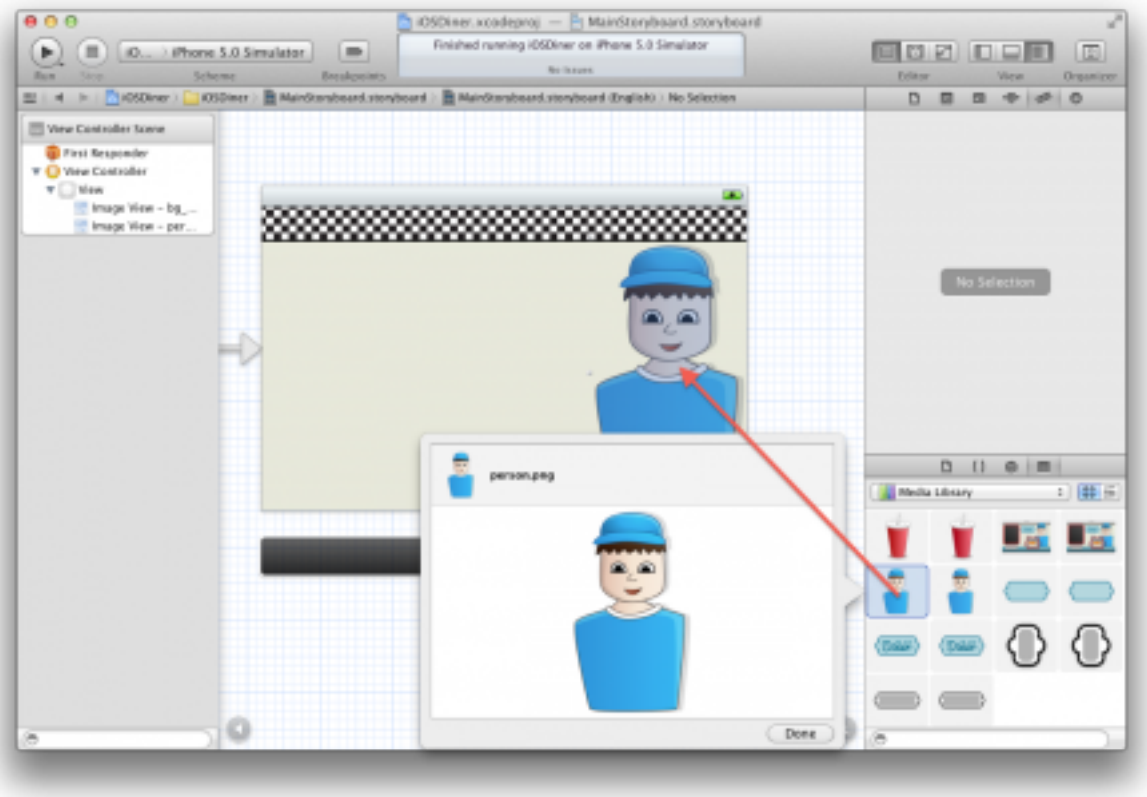


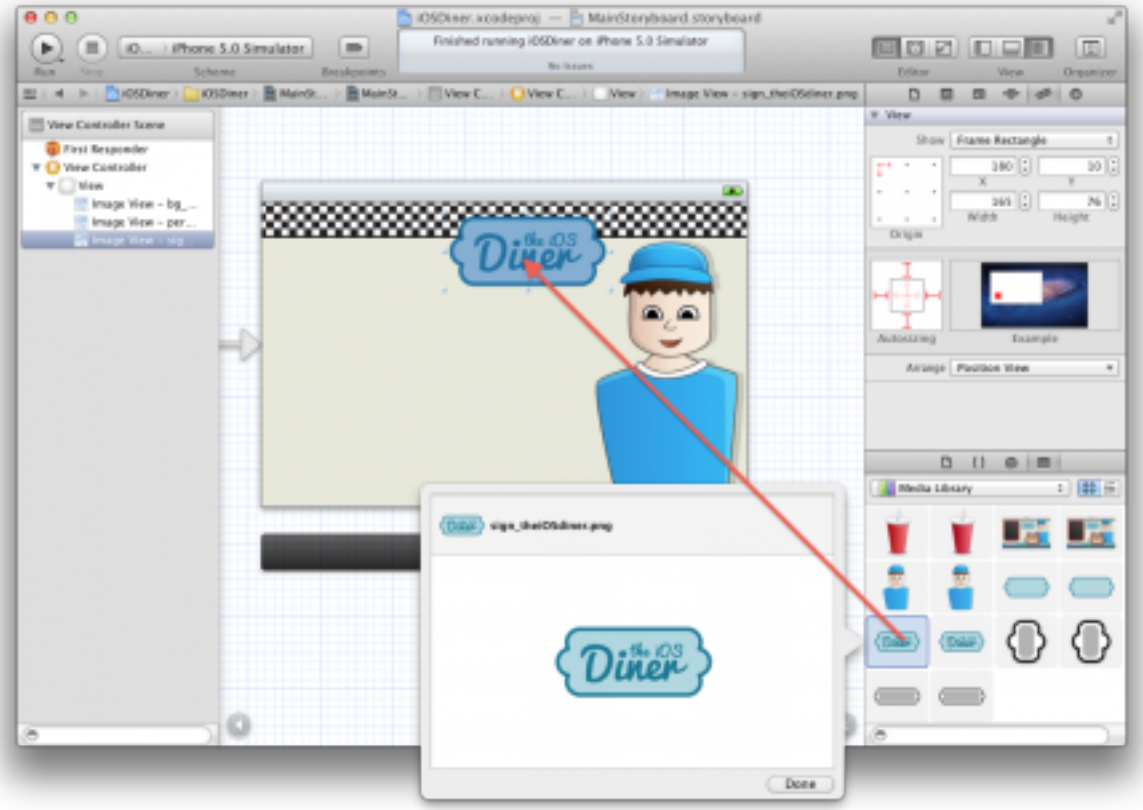
Now do the same with the following images:

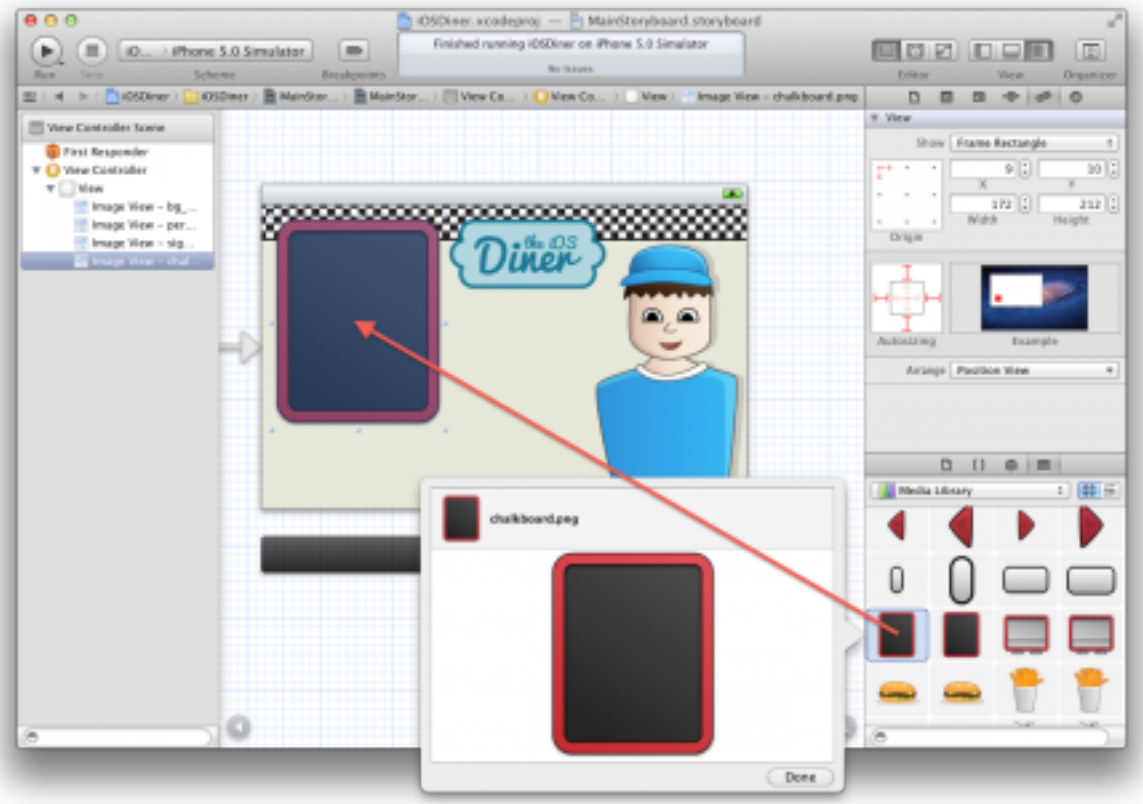
- person.png
- sign_theiOSdiner.png
- chalkboard.png
- bg_counter.png
- total_field.png
- food_box.png

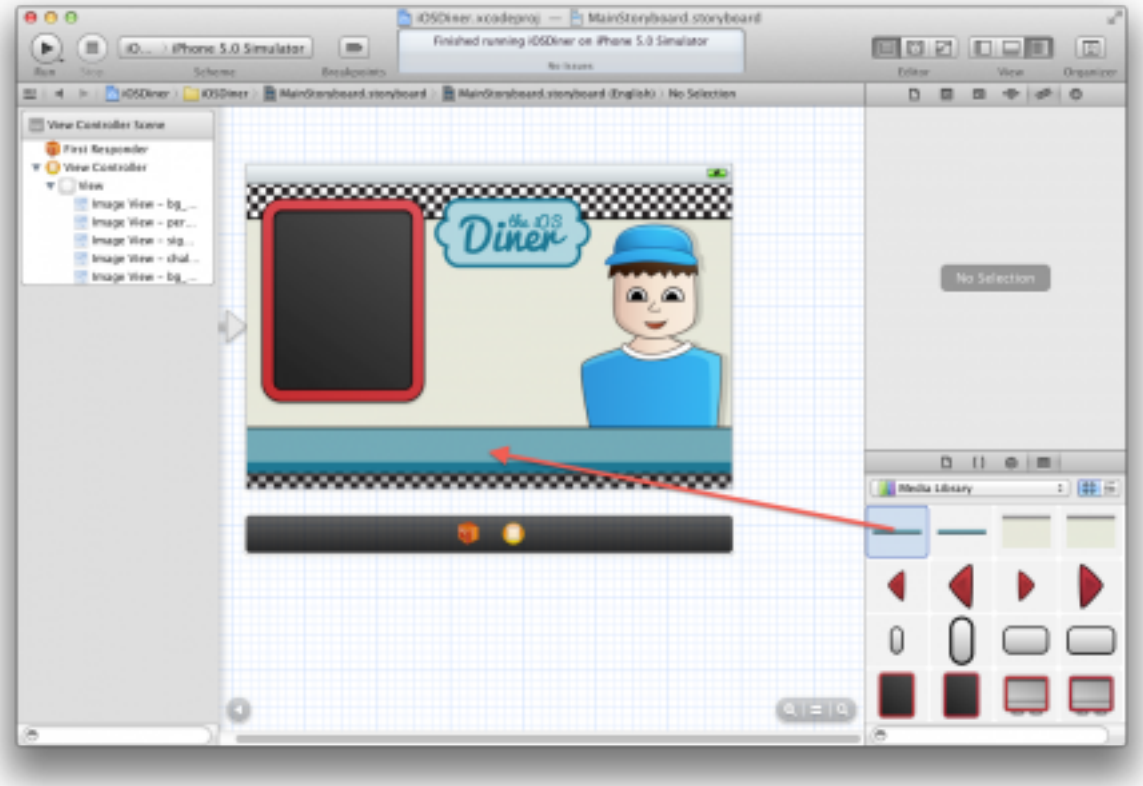
Note that you can make an image view the exact dimensions of an image by selecting the Image View and going to Editor\Size to Fit Conent (or use the keyboard shortcut Command-Equals).

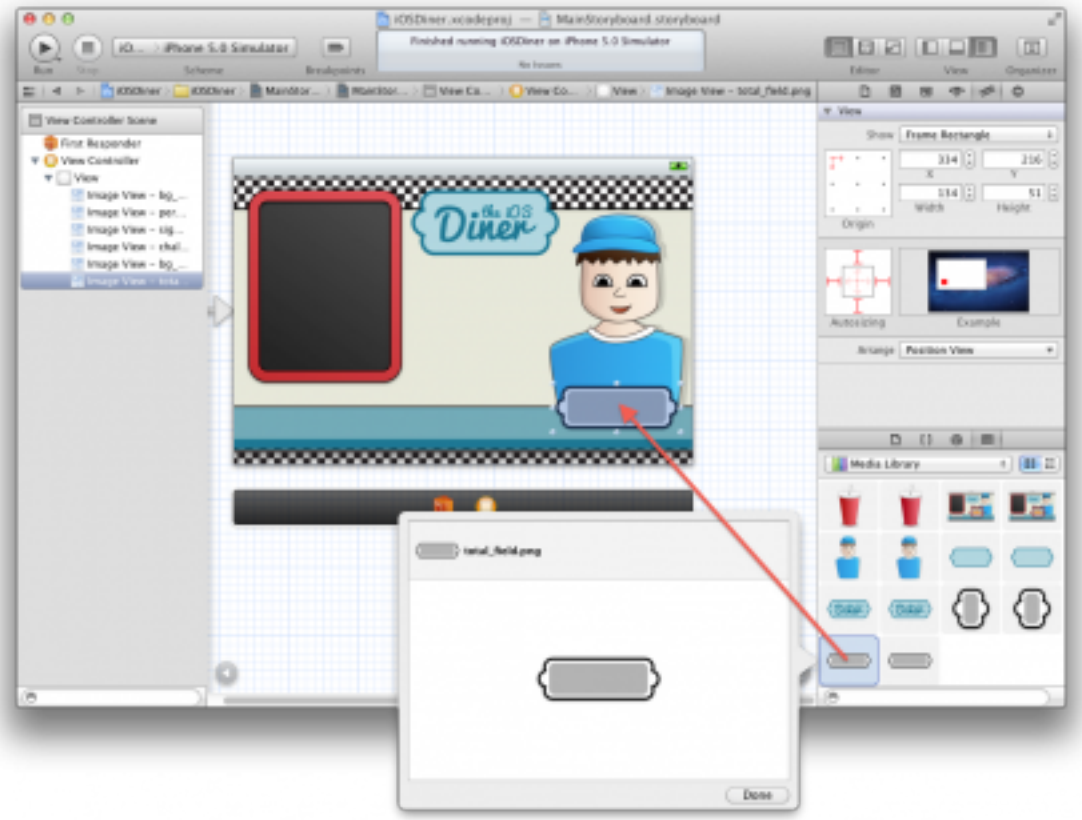
As you drag and position the images, your view should develop as per the images below:

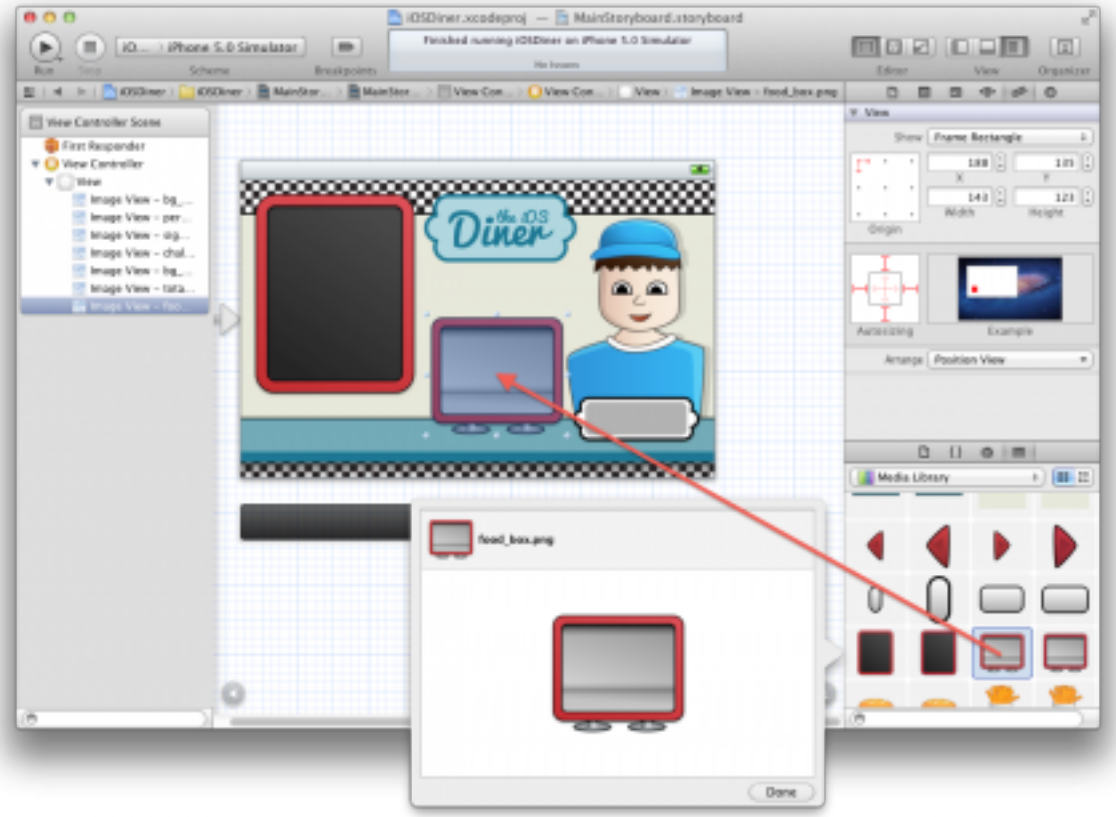



Finished running iOSDiner on iPhone 5.0 Simulator
Media Library
Done


Finished running iOSDiner on iPhone 5.0 Simulator
Media Library
chalkboard.peg
Done


Finished running iOSDiner on iPhone 5.0 Simulator
Diner
No Selection
Media Library


Finished running iOSDiner on iPhone 5.0 Simulator
Diner
Media Library
Done



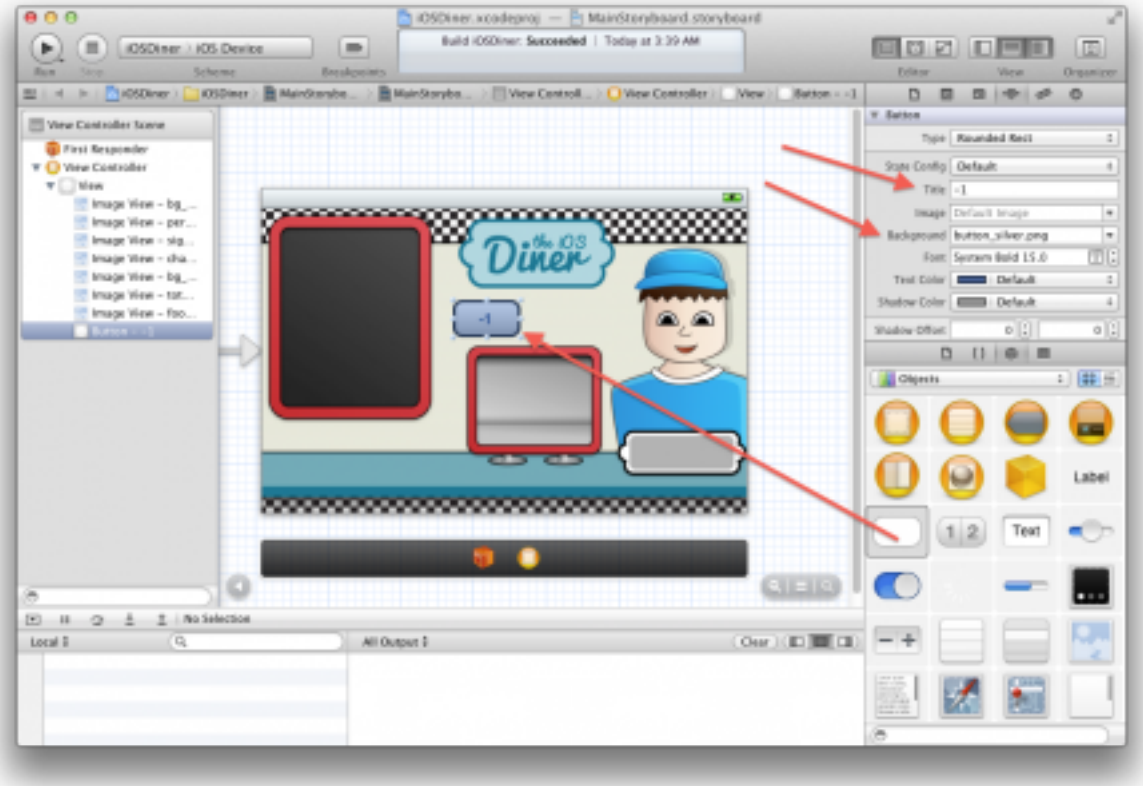
Once you’re done, give the project another Run.

Hey! Almost looking like an app!

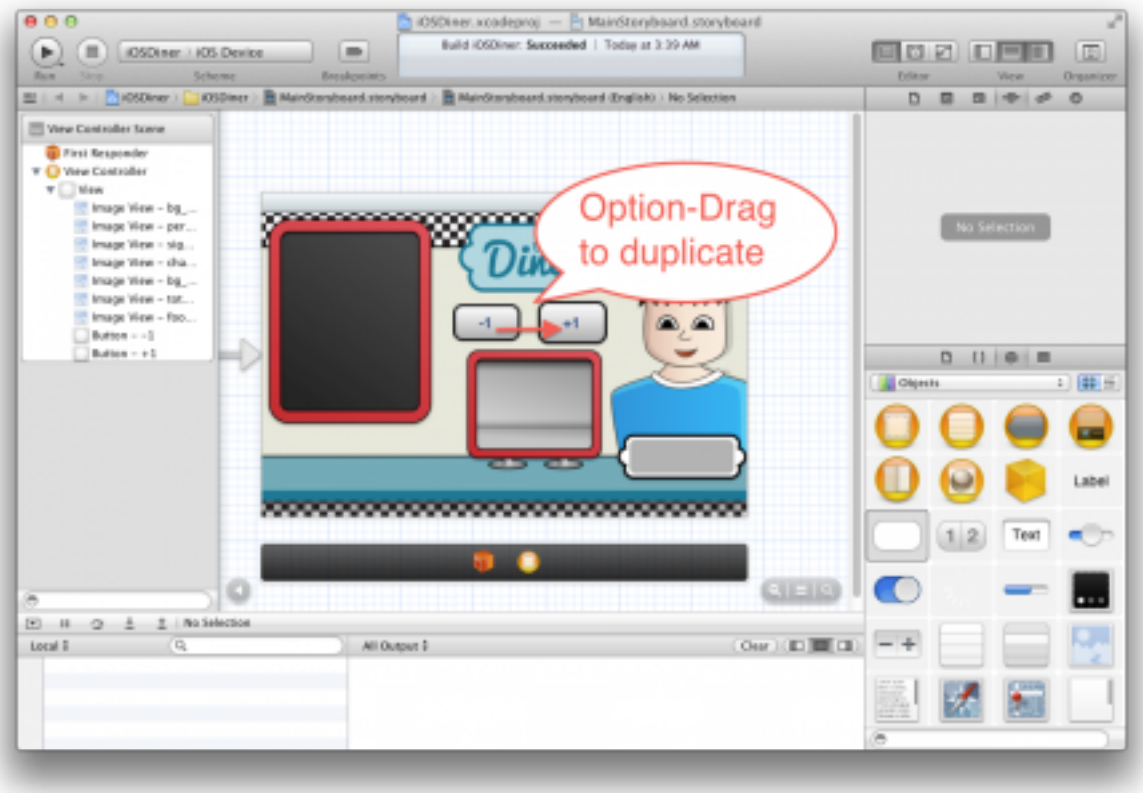
Next, we’re going add the user interface portions. On the Utilities sidebar in Xcode, switch to the Object Library.

Drag a “Round Rect Button” into the middle of the view, above the monitor. Double-click the button you just placed and set the text to “-1”.

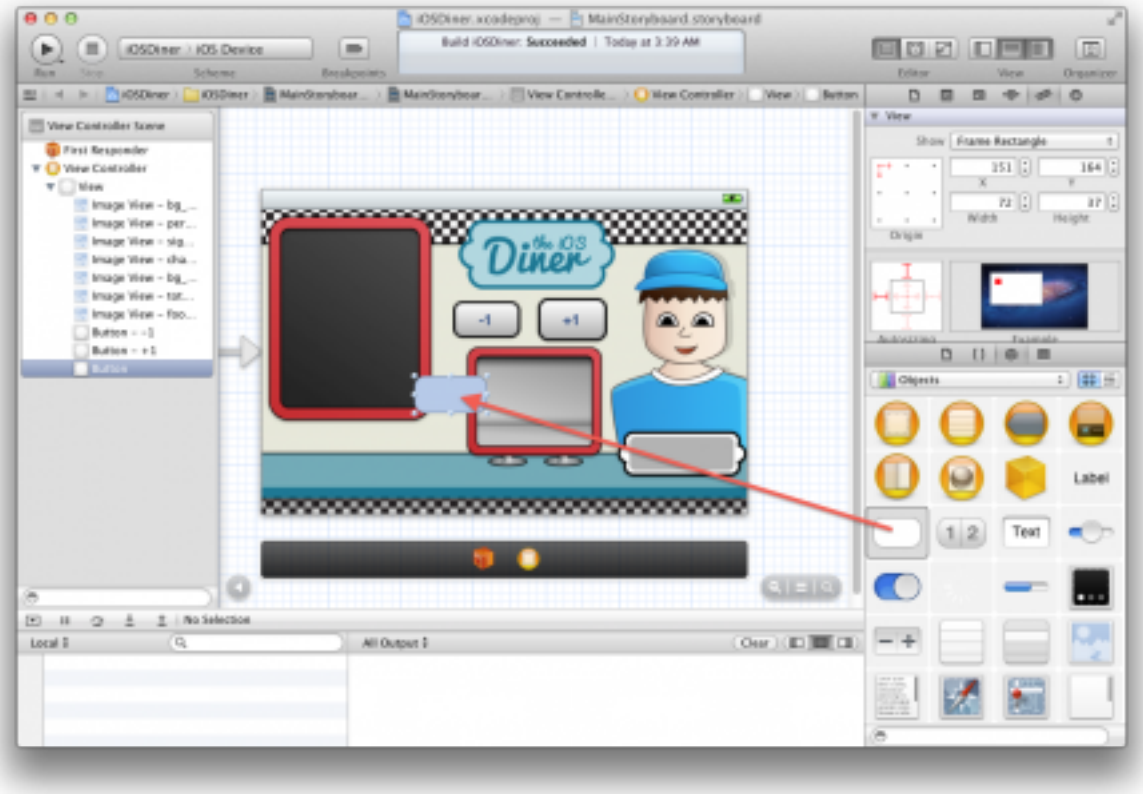
In the Utilities sidebar, select the Attributes Inspector. Make sure that the button you just placed is the currently selected item. Set the Background attribute to “button_silver.png”



Hold down Option/Alt and drag the -1 button to the right. This will create a copy of that object. Double-click this new button and set the text to “+1”.

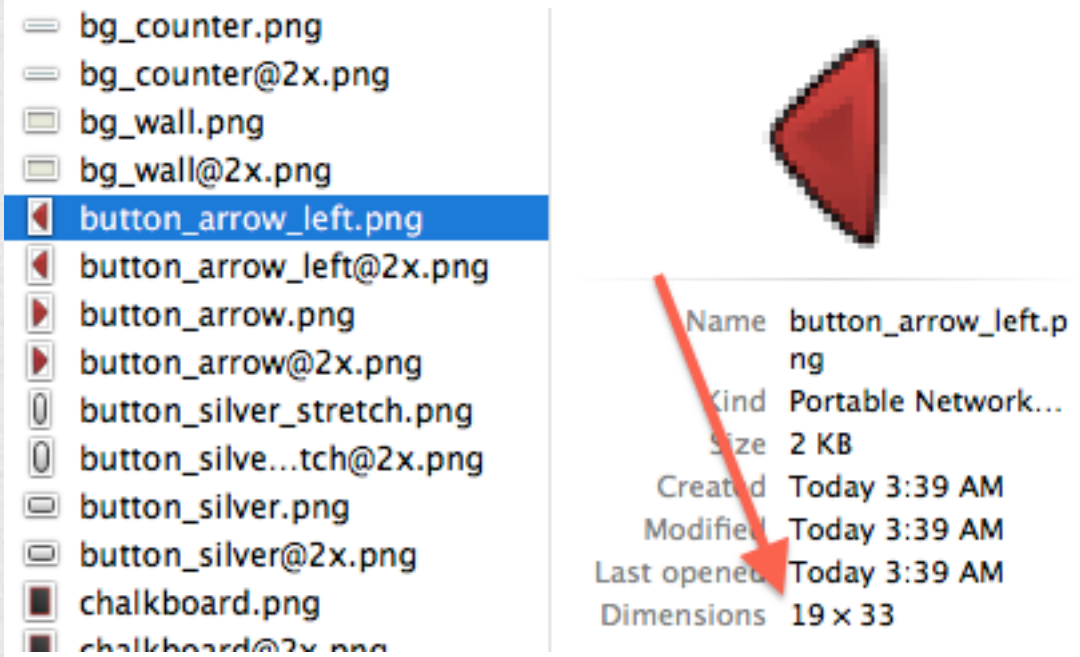


Drag another “Round Rect Button” just to the left of the monitor. Set the Button Type to Custom and the Background attribute of this button to button_arrow_left.png.”

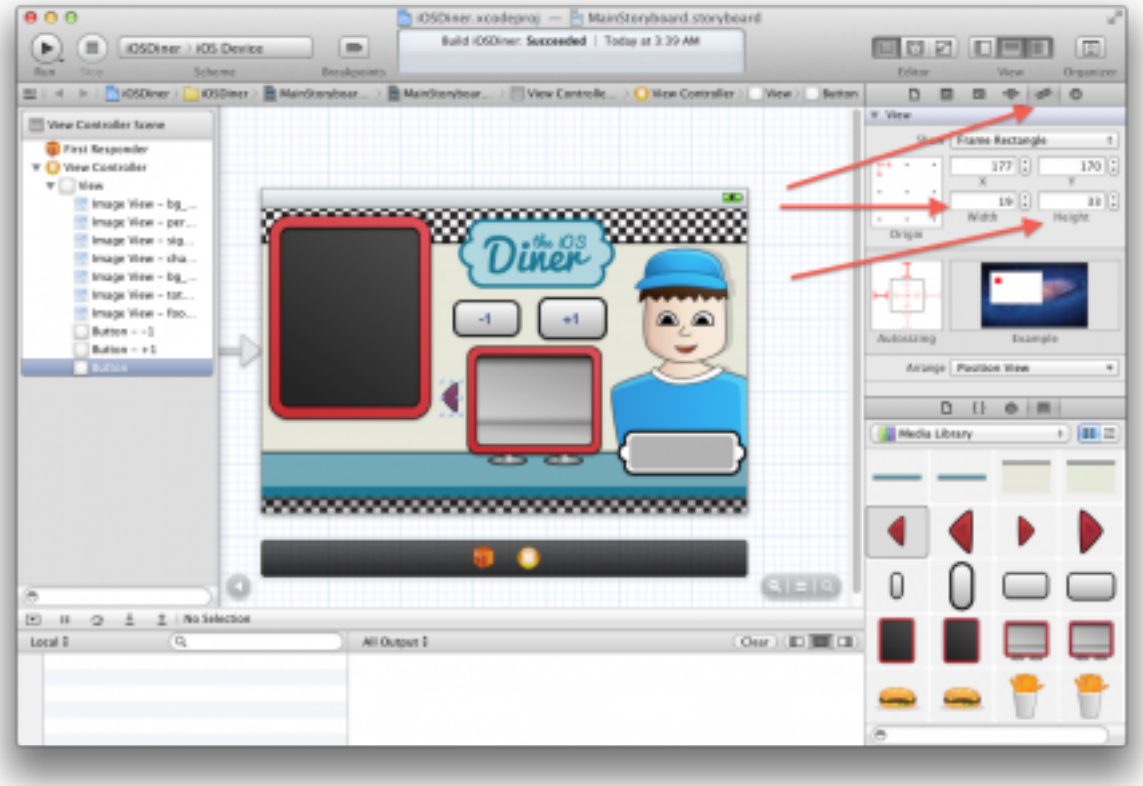


Hmm. That doesn't look quite right. The problem is, when you set the image as the background image for the button, Xcode stretches the image to the size of the button.

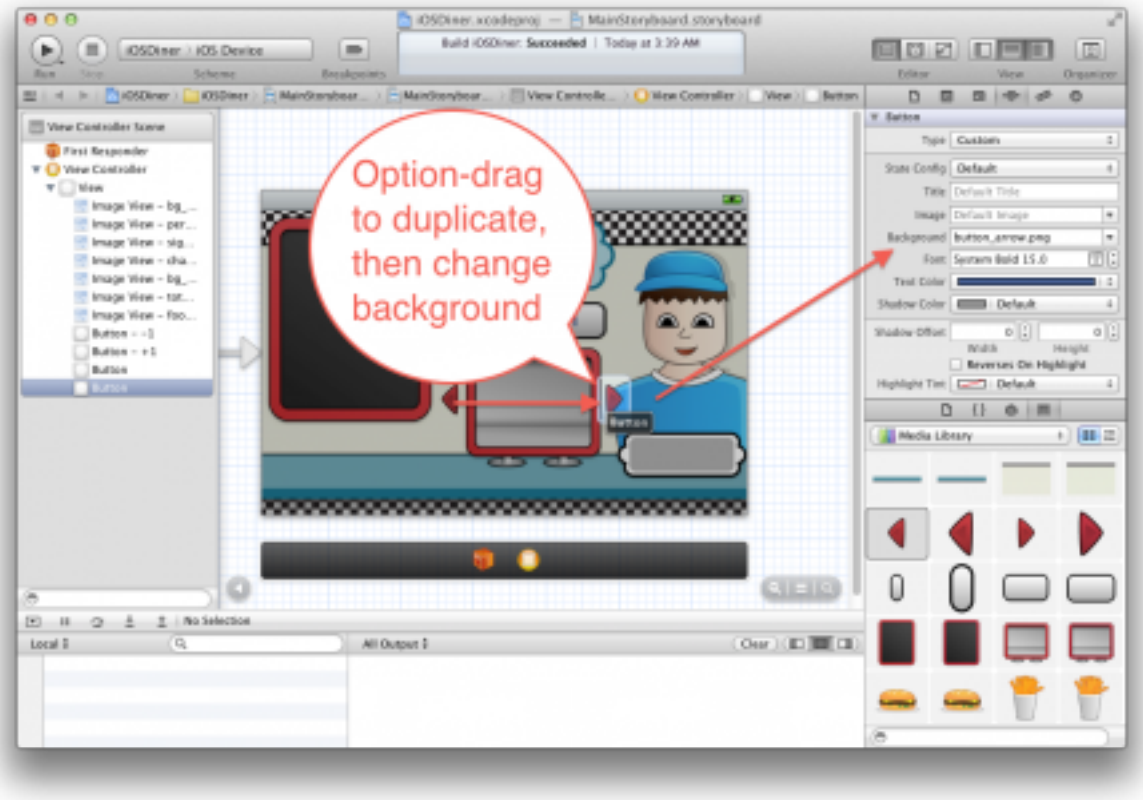
If you look at the file itself, you'll see that the dimensions are 19×33. So all you need to do is set the size of the button to match the size of the image.



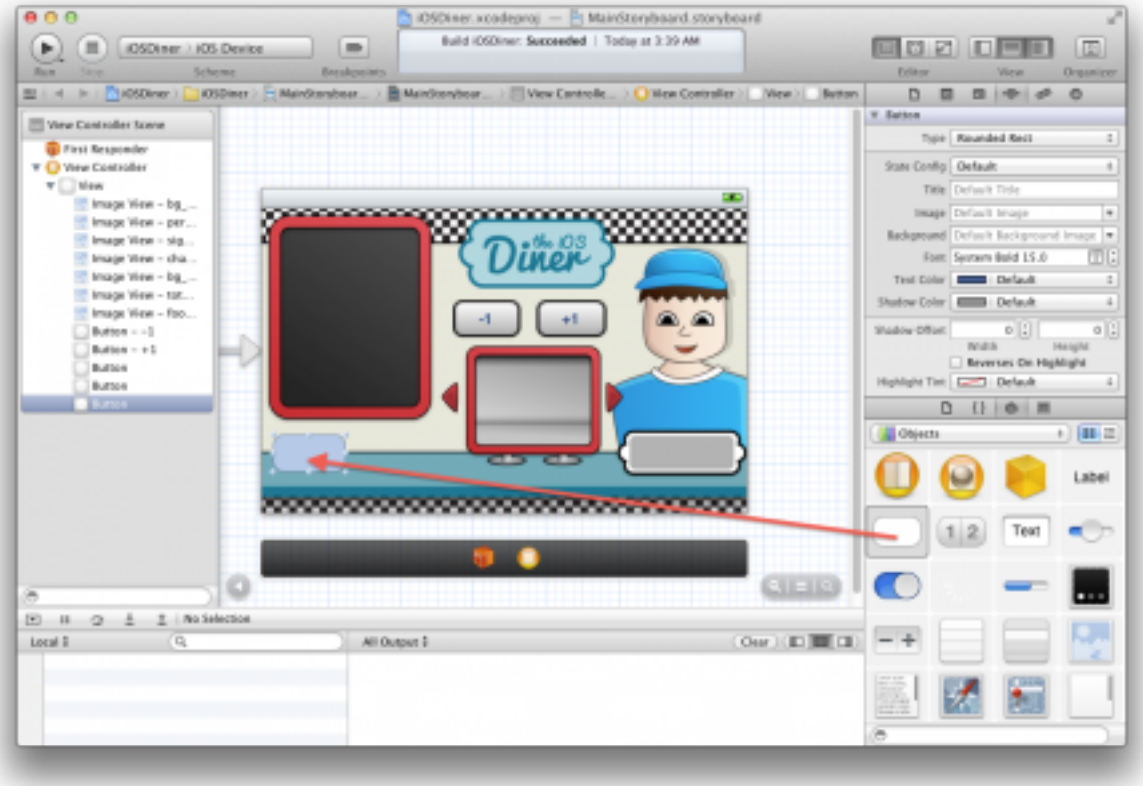
Select the button. Select the Size Inspector, and set the Width and Height to 19 and 33, respectively.



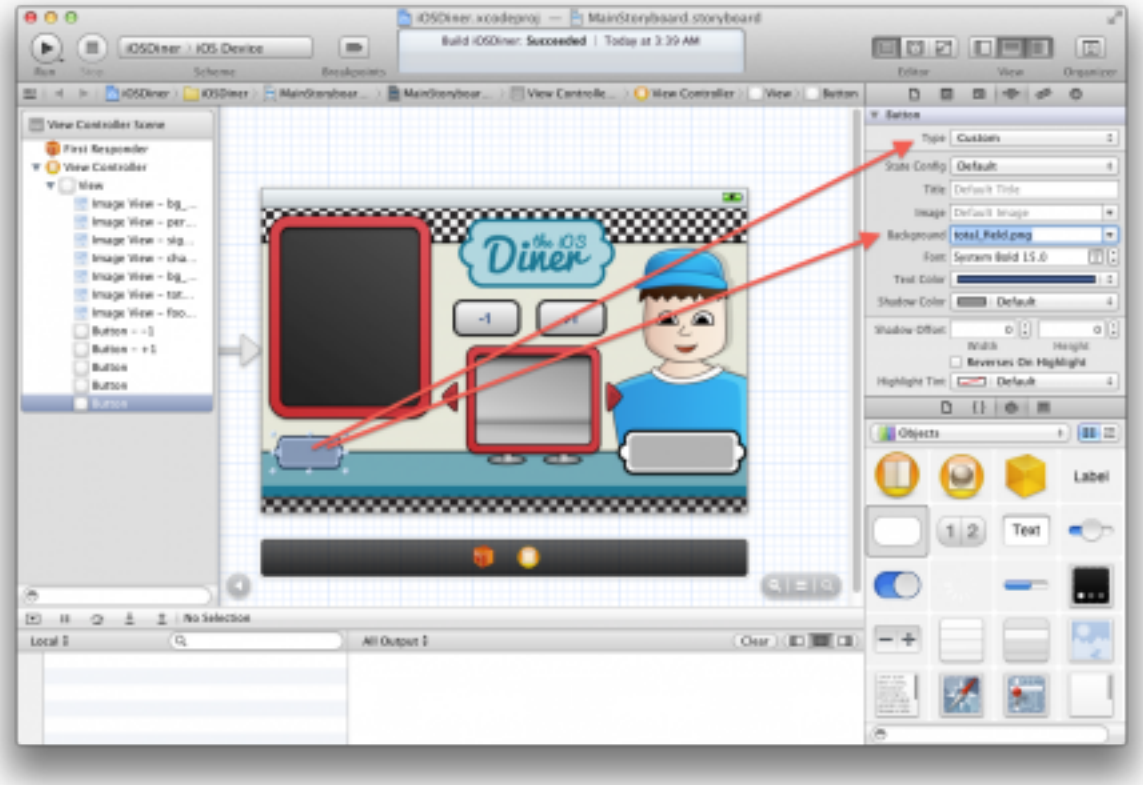
That looks more like it! Now you need a right arrow button on the other side, so press Option/Alt and drag that button onto the other side. Change the background of the new button to button_arrow.png.



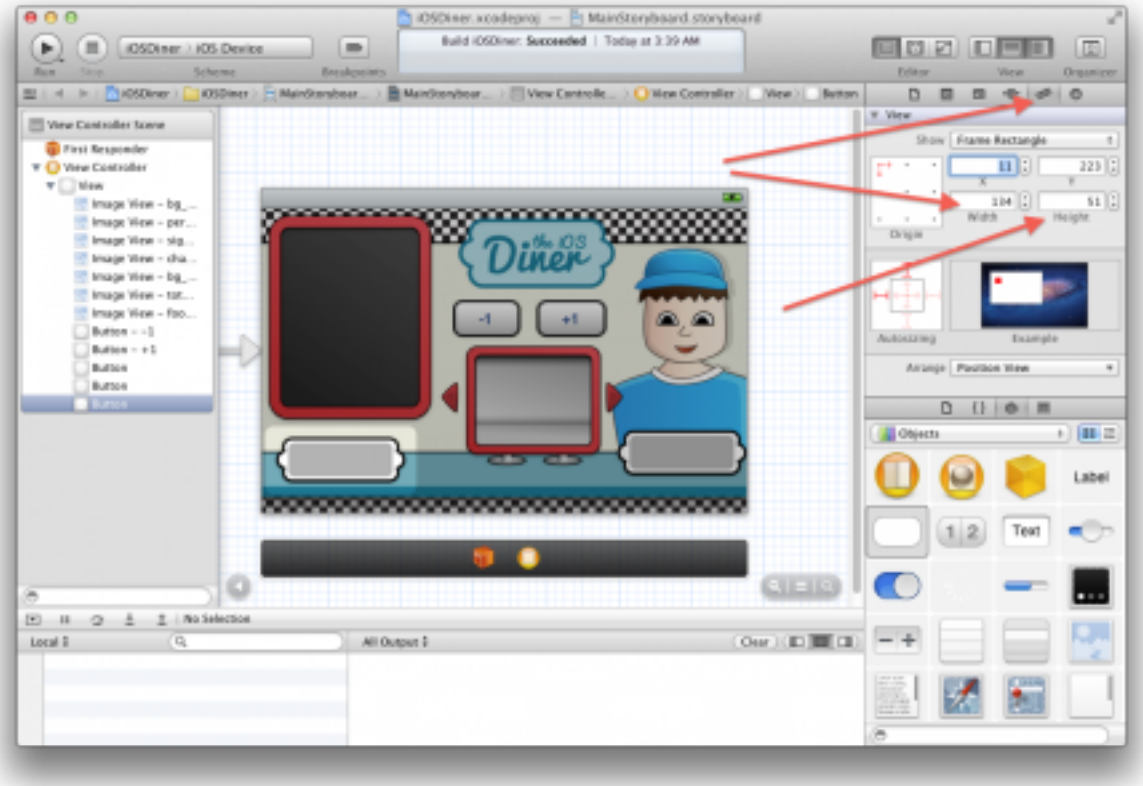
One last button to add! Drag another “Round Rect Button” below the chalkboard.

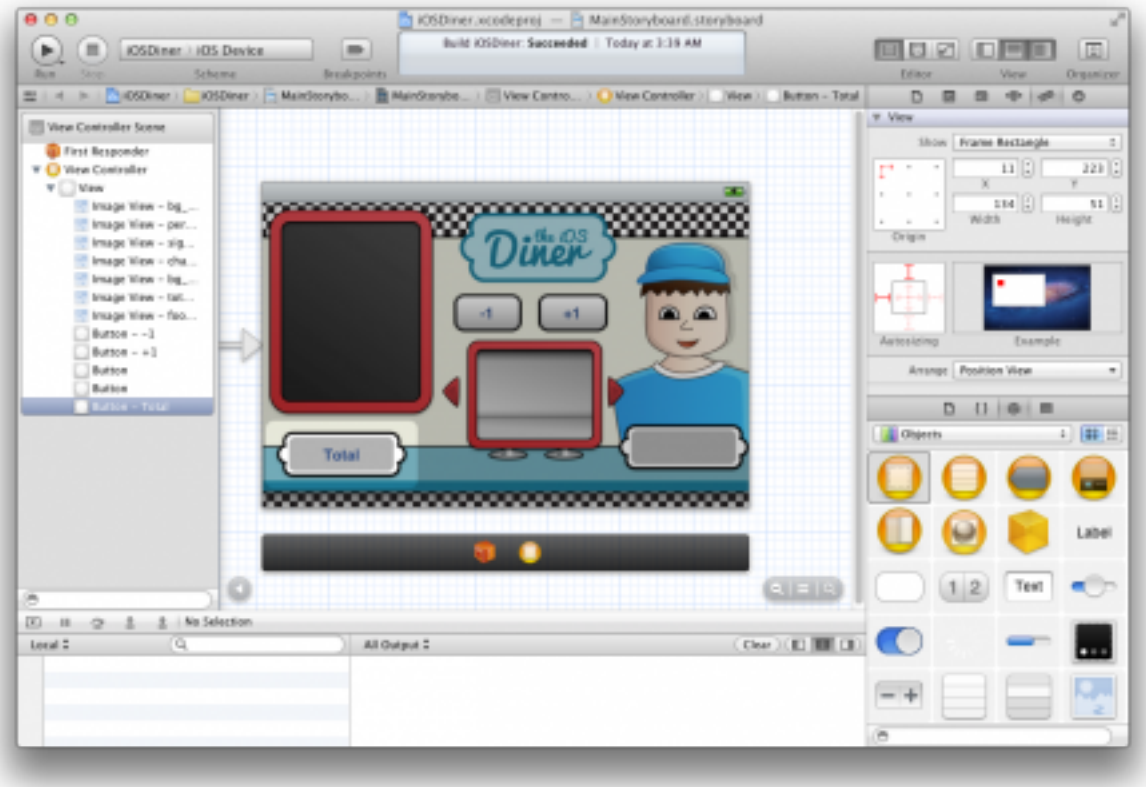


Set the Button Type to Custom and the Background attribute to total_field.png.



Again, we need to set the size to match the image, so select the Size Inspector. Set the Width and Height to 134 and 51, respectively. Then double-click the button and set the text to “Total”.

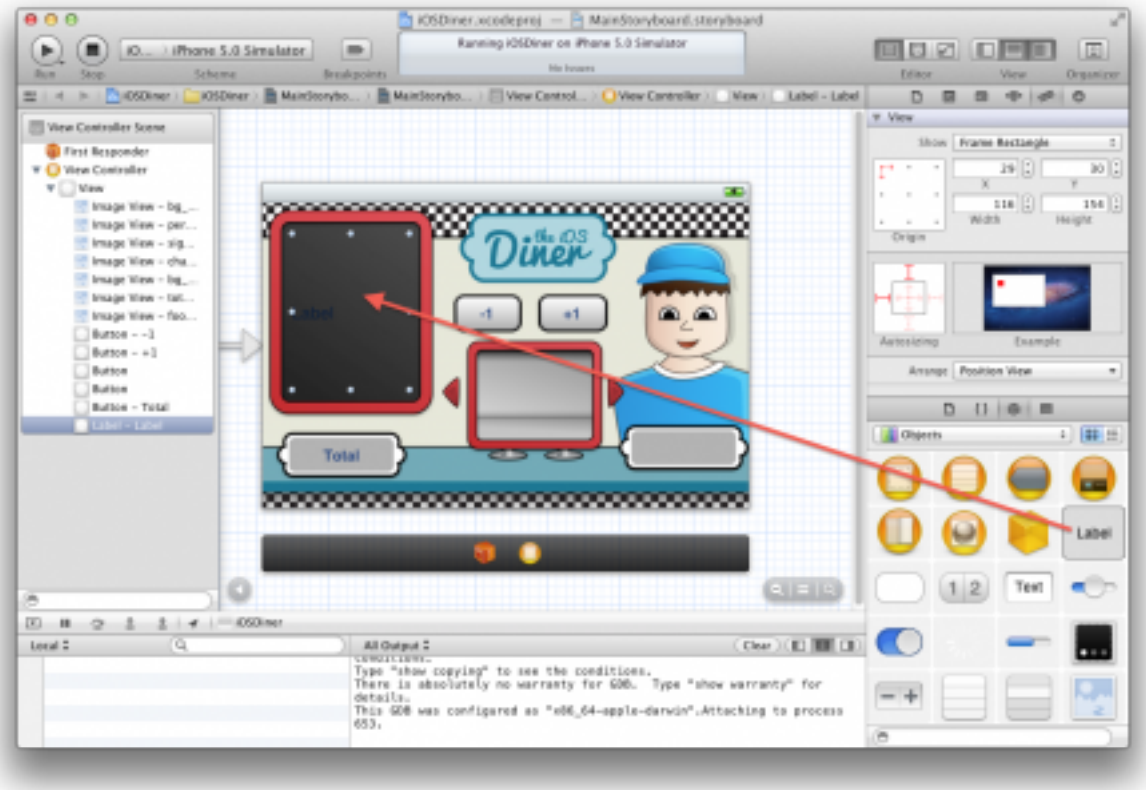




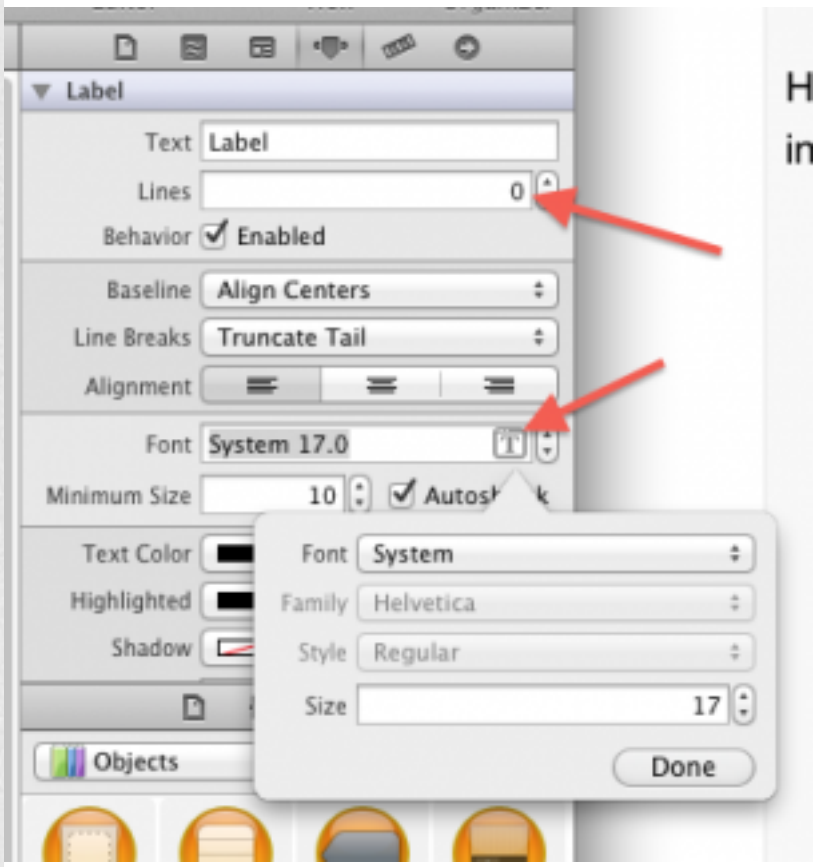
Give the project another Run.

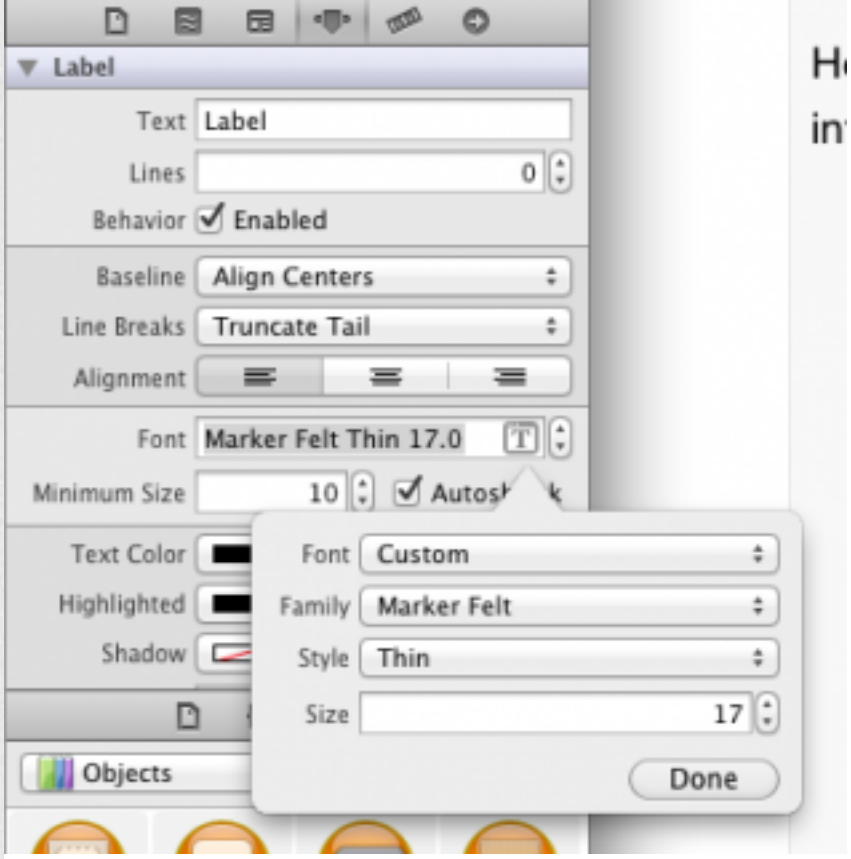
Looks pretty good! The next thing we need to add to the storyboard are the labels and the preview box.

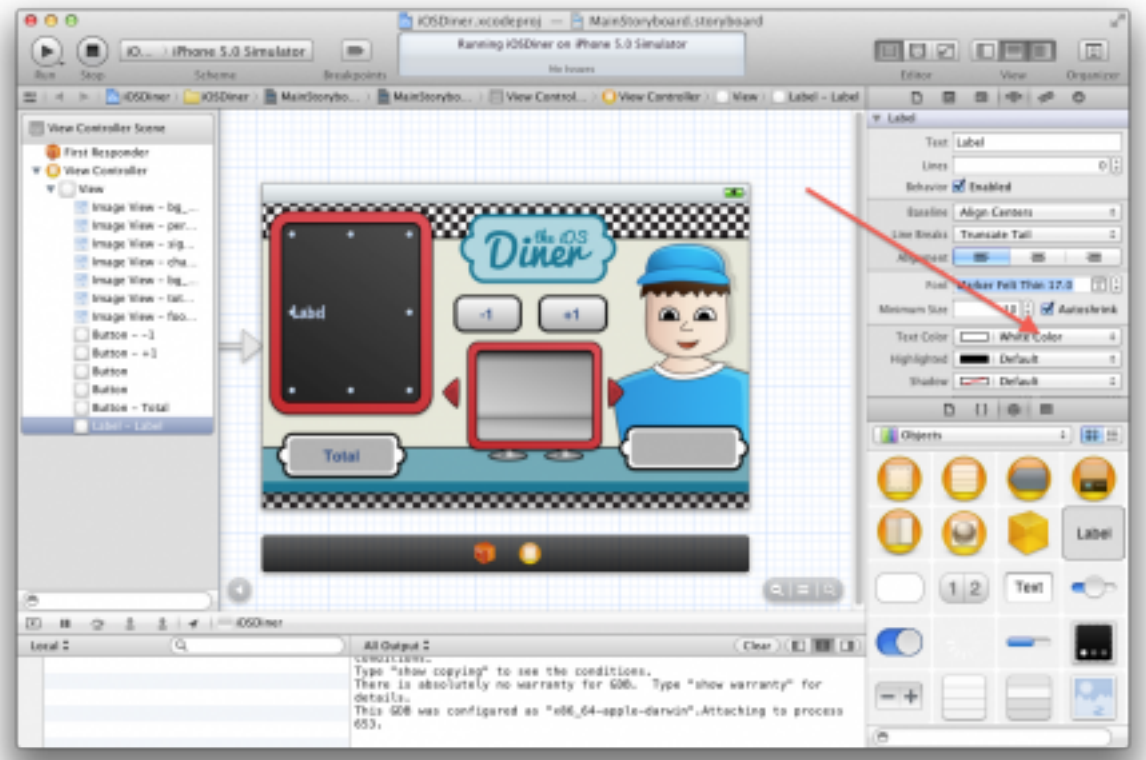
Select the Object Library again. Drag a UILabel onto the chalkboard image. Use the resize handles to make it about the same size as the chalkboard.

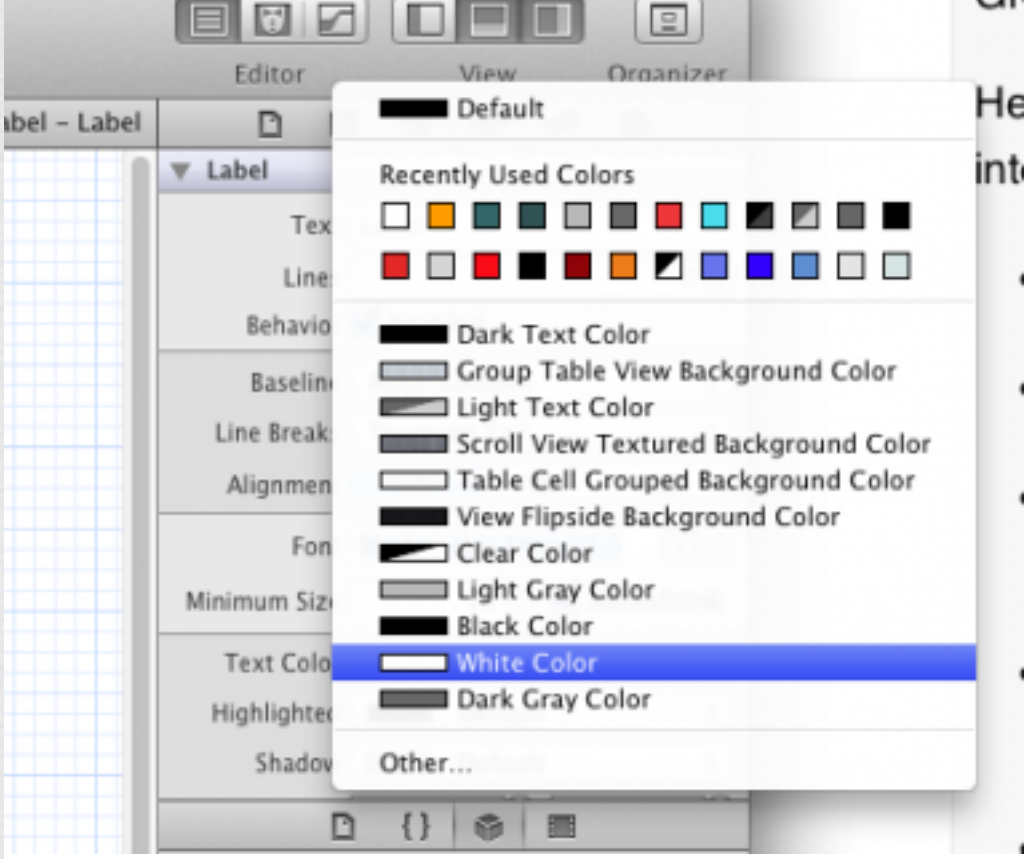


In the Attribute Inspector, set Lines to 0 (which enables multi-line labels), change Text Color to white, and change the Font to Marker Felt 17.0 (click the “T” symbol, select Custom, then Marker Felt). Normally, I find Marker Felt a crime against eyeballs, but in this case it seems to fit.

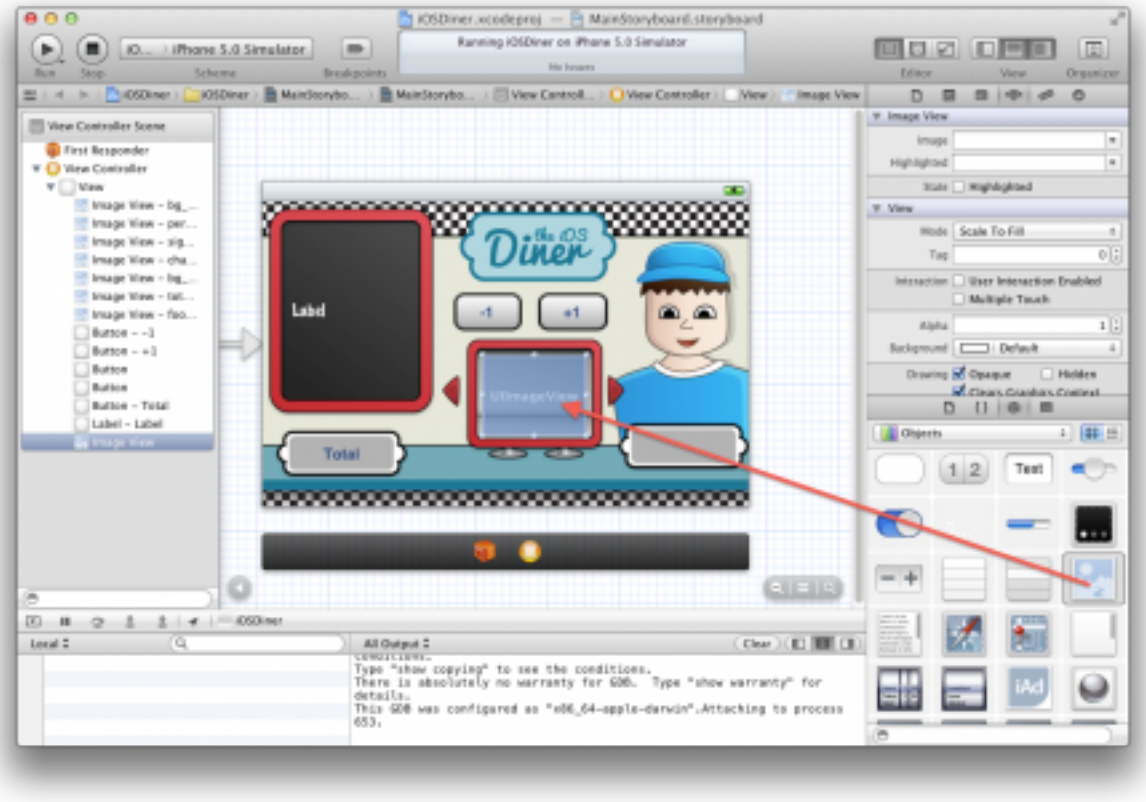



Label
Text Label
Lines 0
Behavior Enabled
Baseline Align Centers
Line Breaks Truncate Tail
Alignment
Font Marker Felt Thin 17.0
Minimum Size 10
Text Color
Highlighted
Shadow
Font Custom
Family Marker Felt
Style Thin
Size 17
Done
Objects

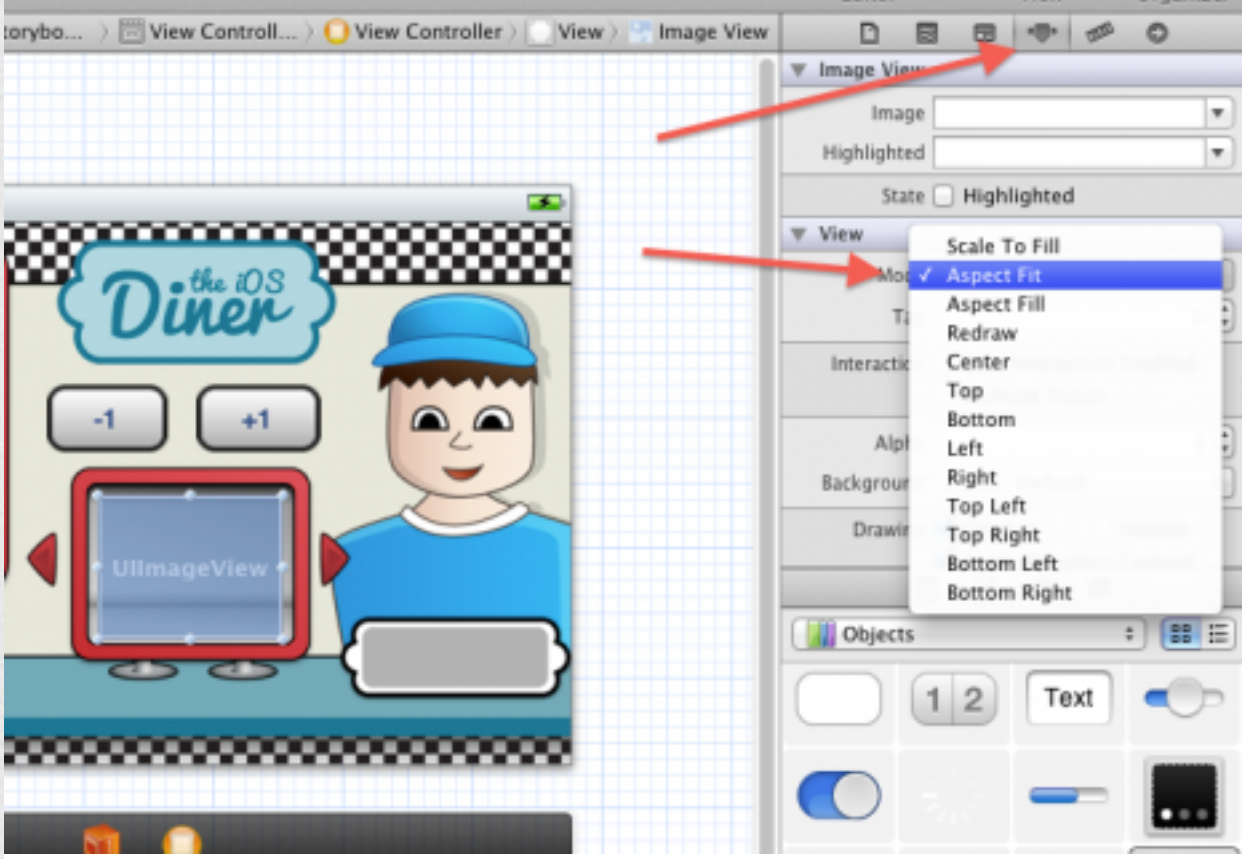

Diner
Label
-1
+1
Total
Label
Text
1 2
Label



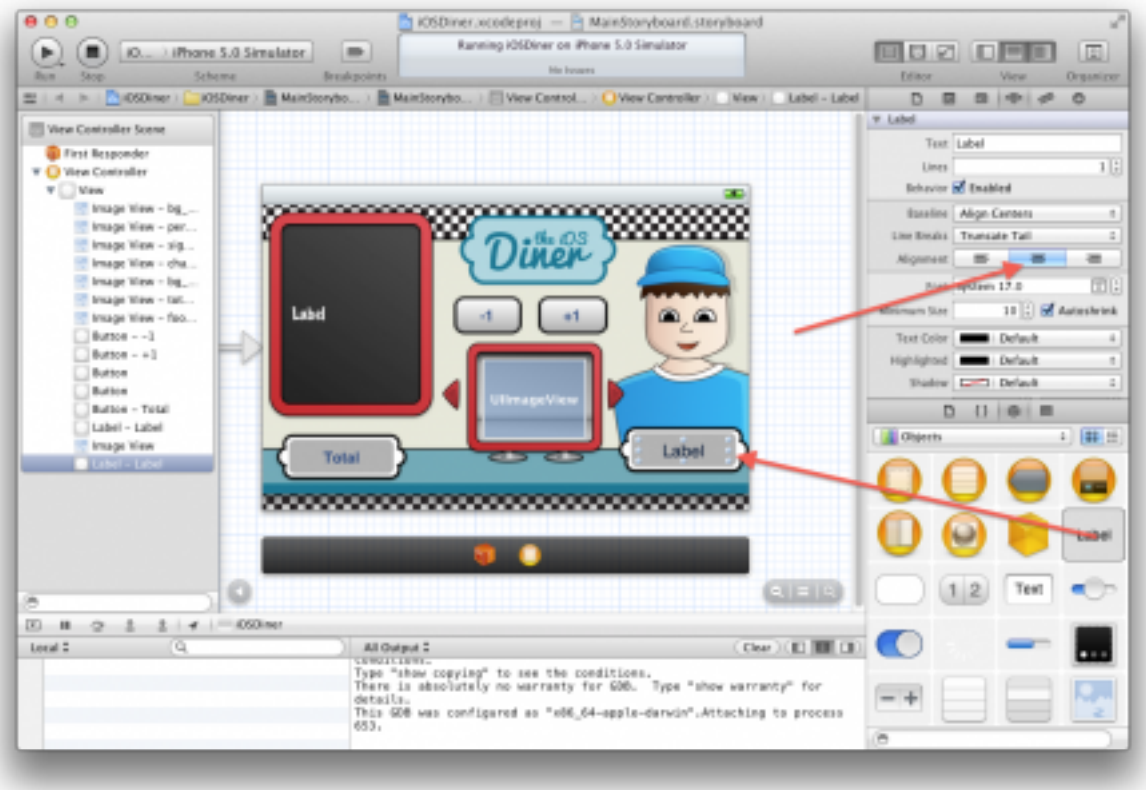
Drag a UIImageView over the monitor in the middle and resize it to match the monitor size.



In the Attributes Inspector, change the Mode to Aspect Fit.



Drag another UILabel over the sign in the lower right. Resize it to be about the same as the grey region, and set the Alignment to Centered.



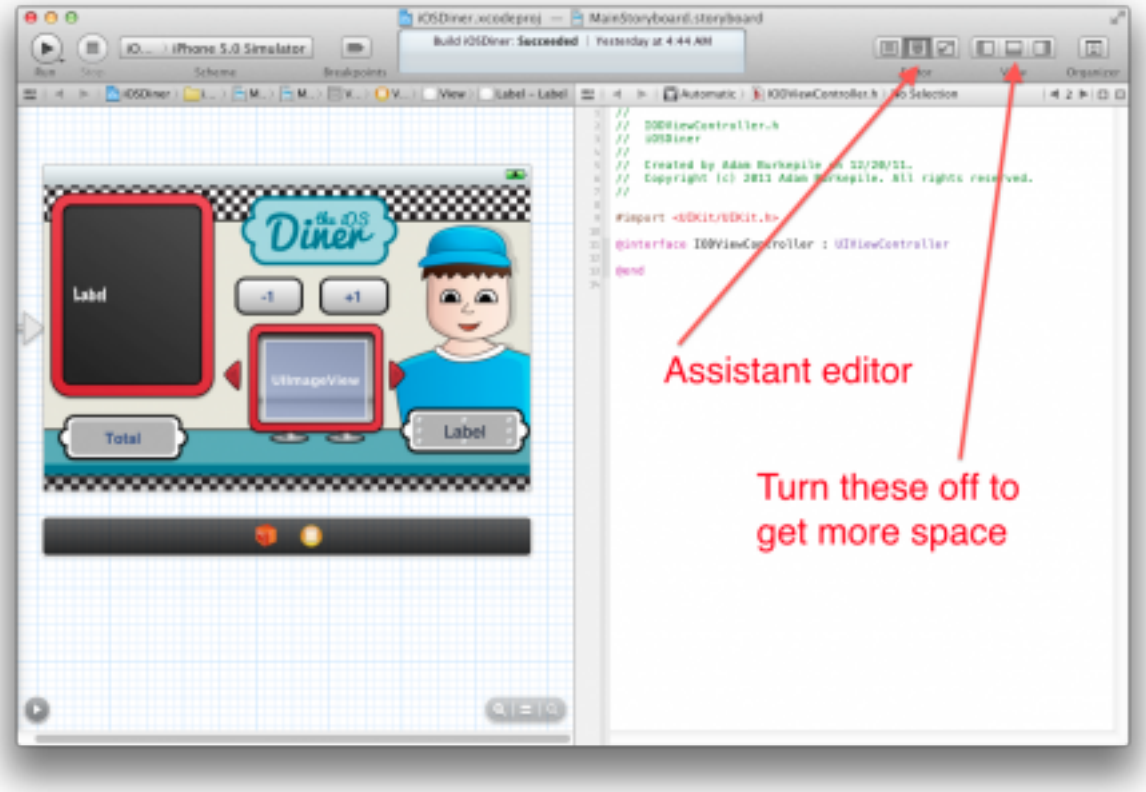
Setting Up IBOutlets and IBActions

Now we need to set up the connection between the user interface we just created and our code. This is where IBOutlets and IBActions come in. The IB part here refers to Interface Builder which is used to create user interfaces in Xcode.

- An **IBOutlet** is basically a connection between a user interface element (for instance a label or a button) and the reference to that element in our code.
- An **IBAction** is an action (or a method, if you prefer) in our code which can be hooked up to a specific event (tapping a button, for example) in the interface we designed.

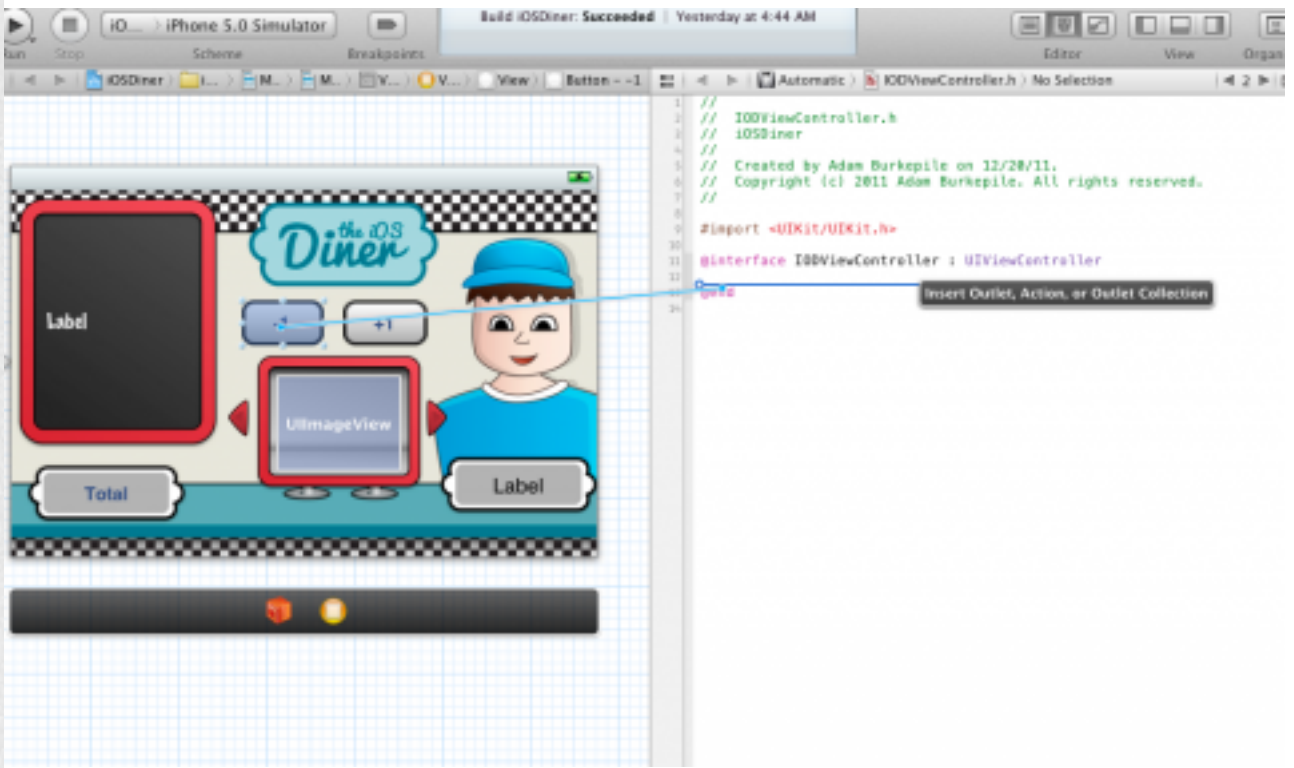
So let's write some code to hook the various UI elements via IBOutlets and IBActions.

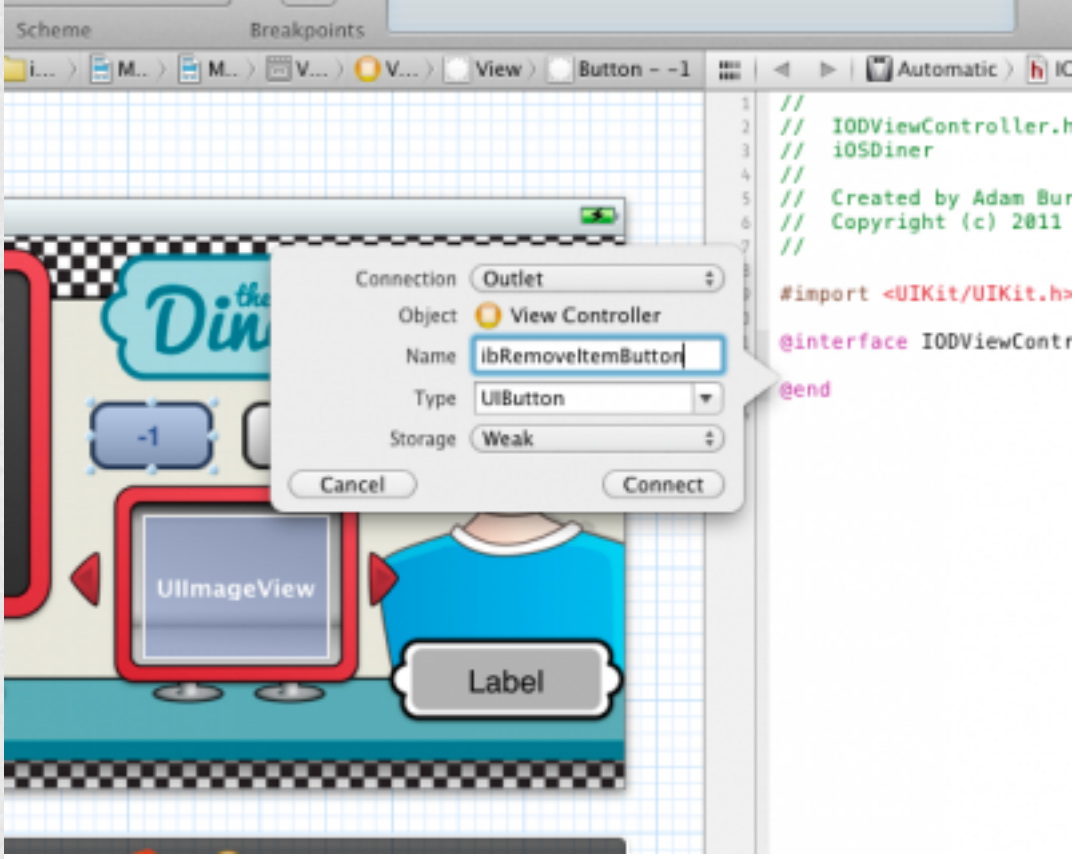
Close the Utilities sidebar and open the Assistant editor. Depending on how you have the assistant editor set to display, your screen might (or might not) look like the screenshot below. If you want to change how the Assistant editor is displayed, you can do so via the View – Assistant Editor menu option.



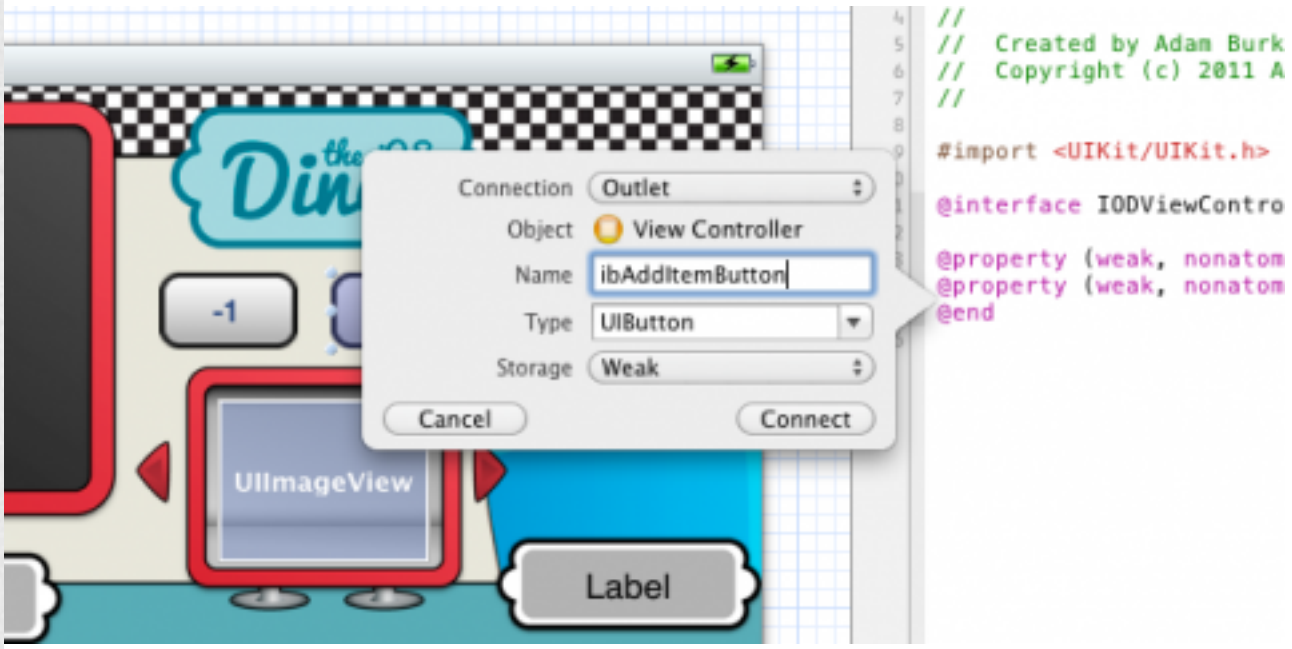
Let's start with the buttons. Select the "-1" button, hold down the Control key and and drag into the code in the Editor view. This will automate the creation of an IBOutlet for the button.

For this object, all you need to do is name it. I like to prefix all my outlets with "ib" so it makes it easier to find all my outlets in Xcode autocomplete. Name this object "ibRemoveItemButton" and click Connect.

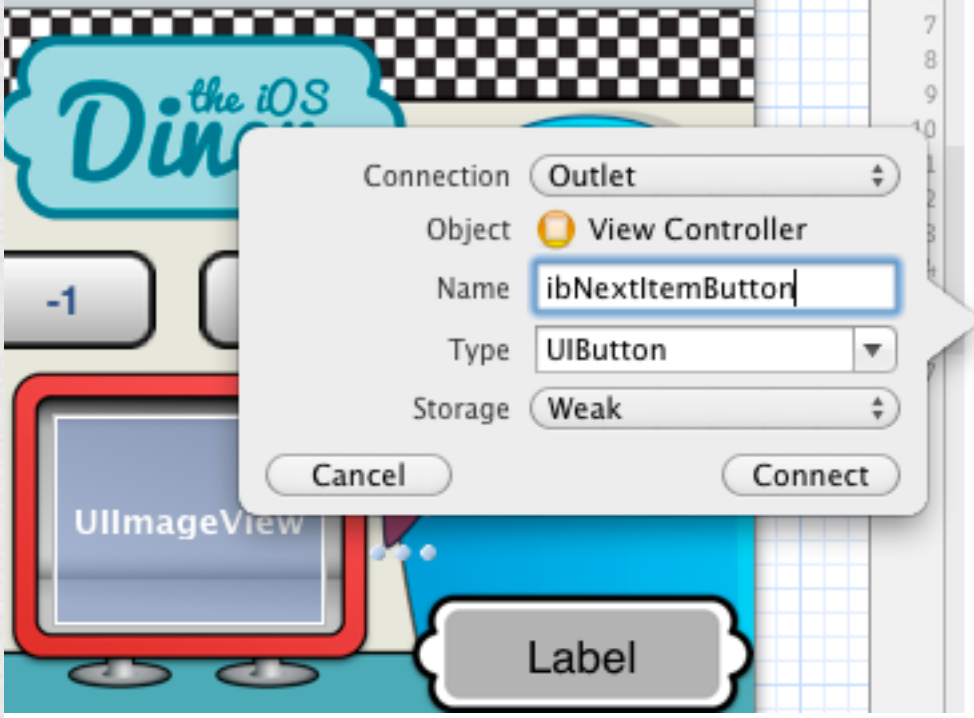




Now do the same for the “+1” button, and name it “ibAddItemButton.”



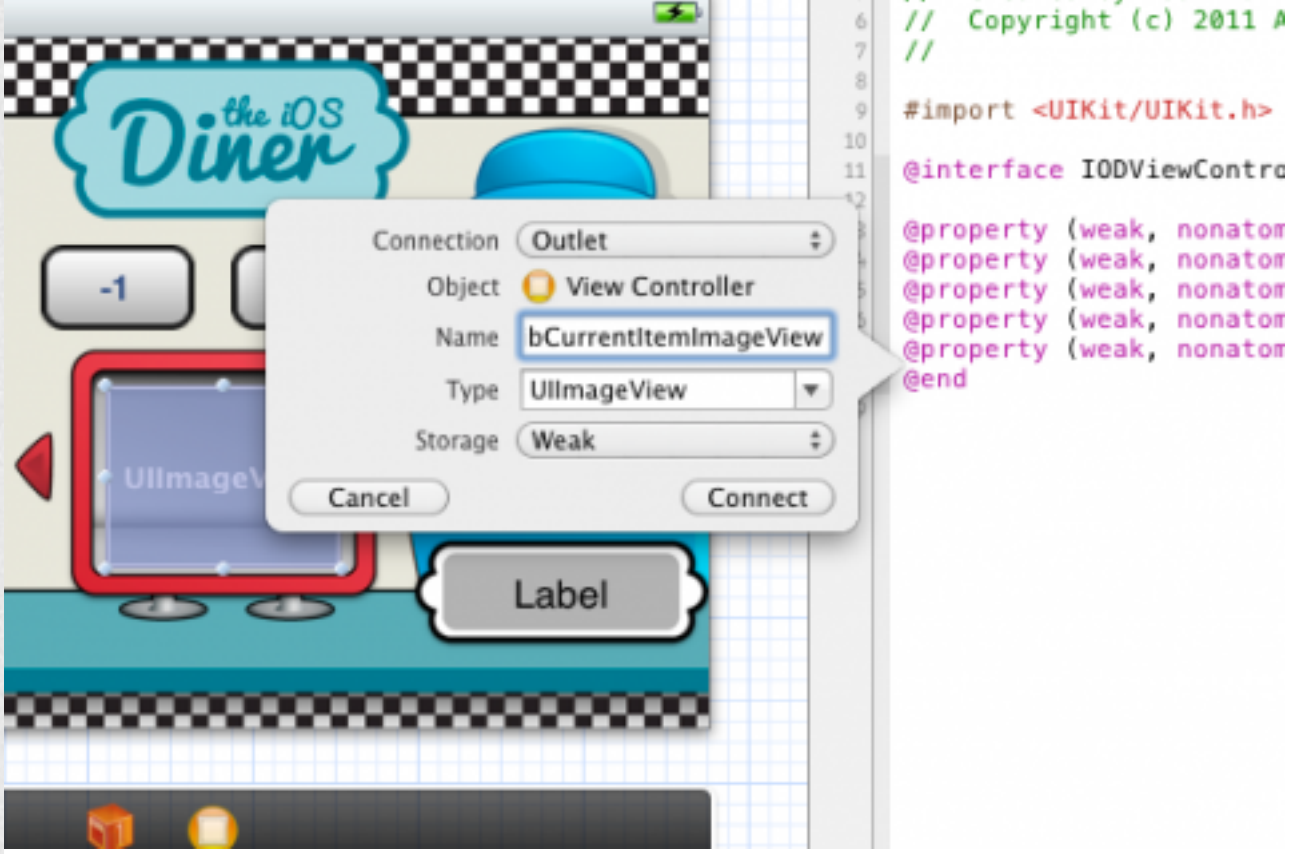
Next, set up outlets for the red arrow buttons. Name them “ibPreviousItemButton” and “ibNextItemButton.”

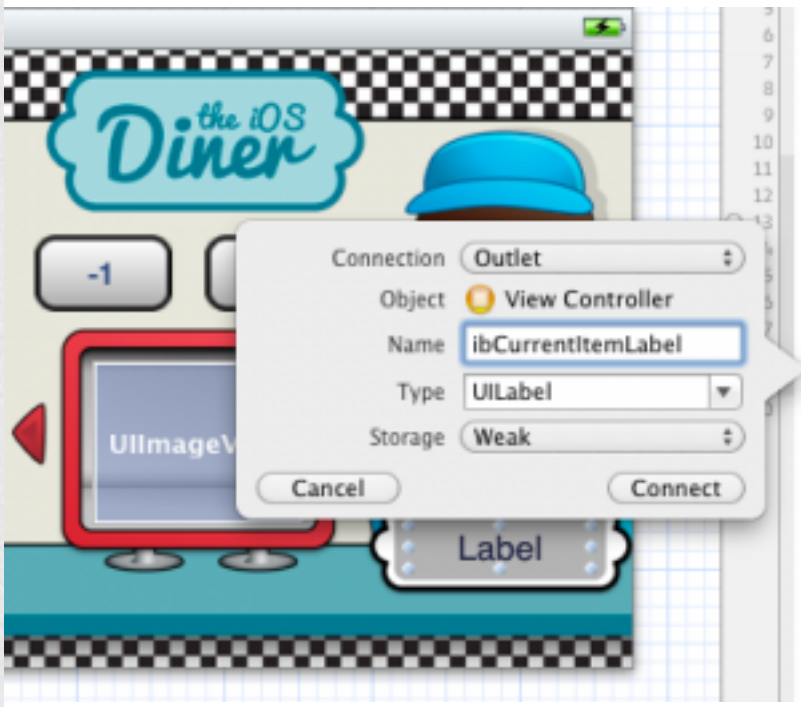


Set up an outlet for the total button named “ibTotalOrderButton.”

Now set up outlets for the labels and the image view. From left to right, name them:

- ibChalkboardLabel
- ibCurrentItemImageView
- ibCurrentItemLabel





Your IODViewController.h should now look like this:

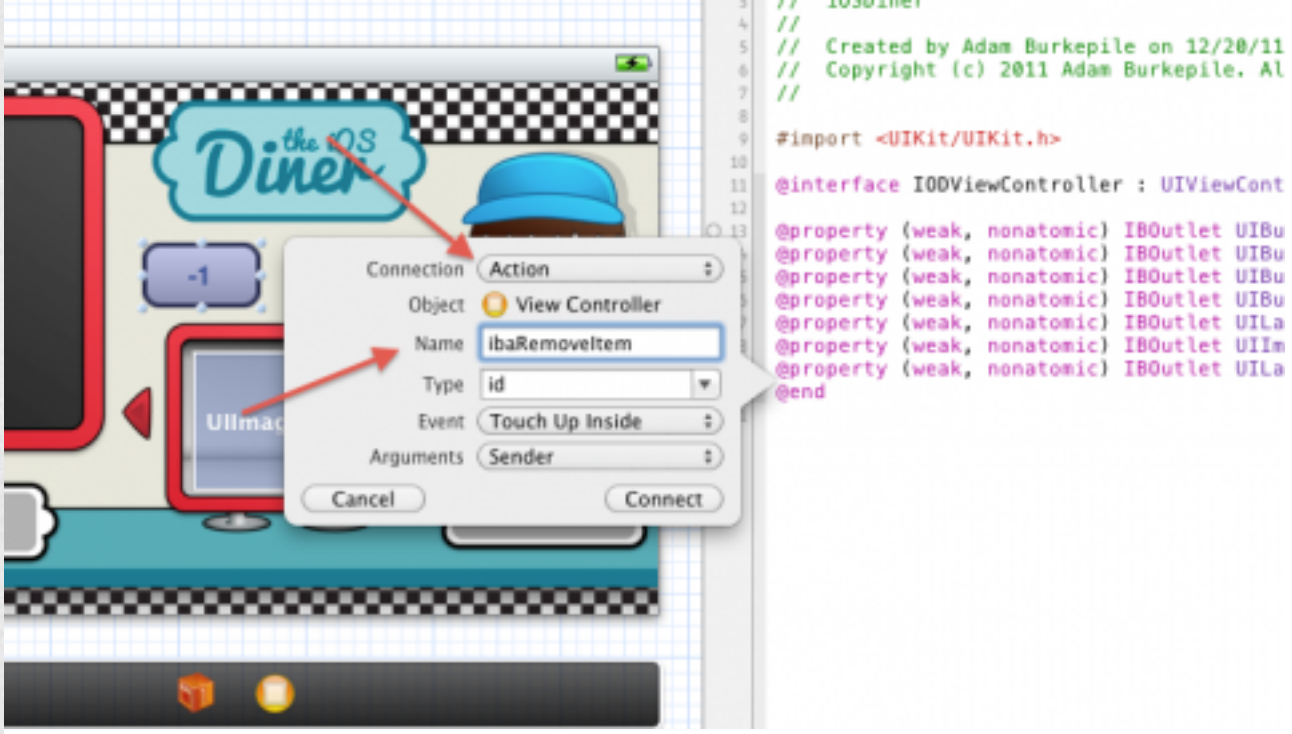
```
@interface IODViewController : UIViewController

@property (weak, nonatomic) IBOutlet UIButton *ibRemoveItemButton;
@property (weak, nonatomic) IBOutlet UIButton *ibAddItemButton;
@property (weak, nonatomic) IBOutlet UIButton *ibPreviousItemButton;
@property (weak, nonatomic) IBOutlet UIButton *ibNextItemButton;
@property (weak, nonatomic) IBOutlet UILabel *ibChalkboardLabel;
@property (weak, nonatomic) IBOutlet UIImageView *ibCurrentItemImageView;
@property (weak, nonatomic) IBOutlet UILabel *ibCurrentItemLabel;
@end
```

Finally, we are going to add the IBActions for the UIButtons. These are the methods that get called in response to certain events (Touch Up Inside, Touch Up Outside, Touch Cancel, etc). With buttons, most of the time we're using Touch Up Inside.

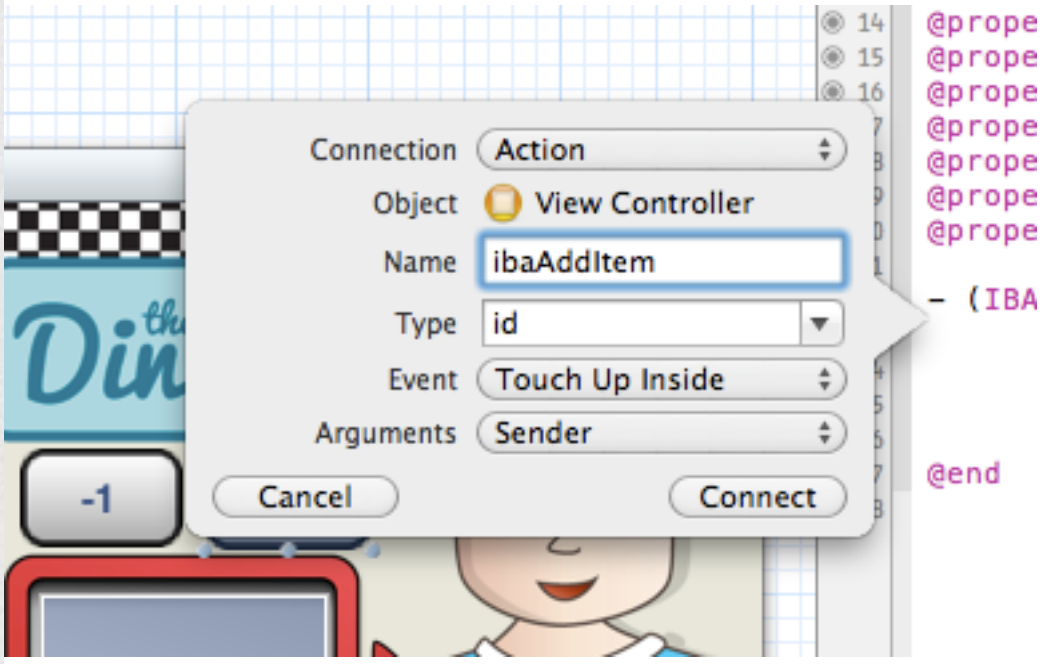
Select the "-1" button again. Control-drag over to the .h file again.

Be sure to remember to change the Connection type to Action.

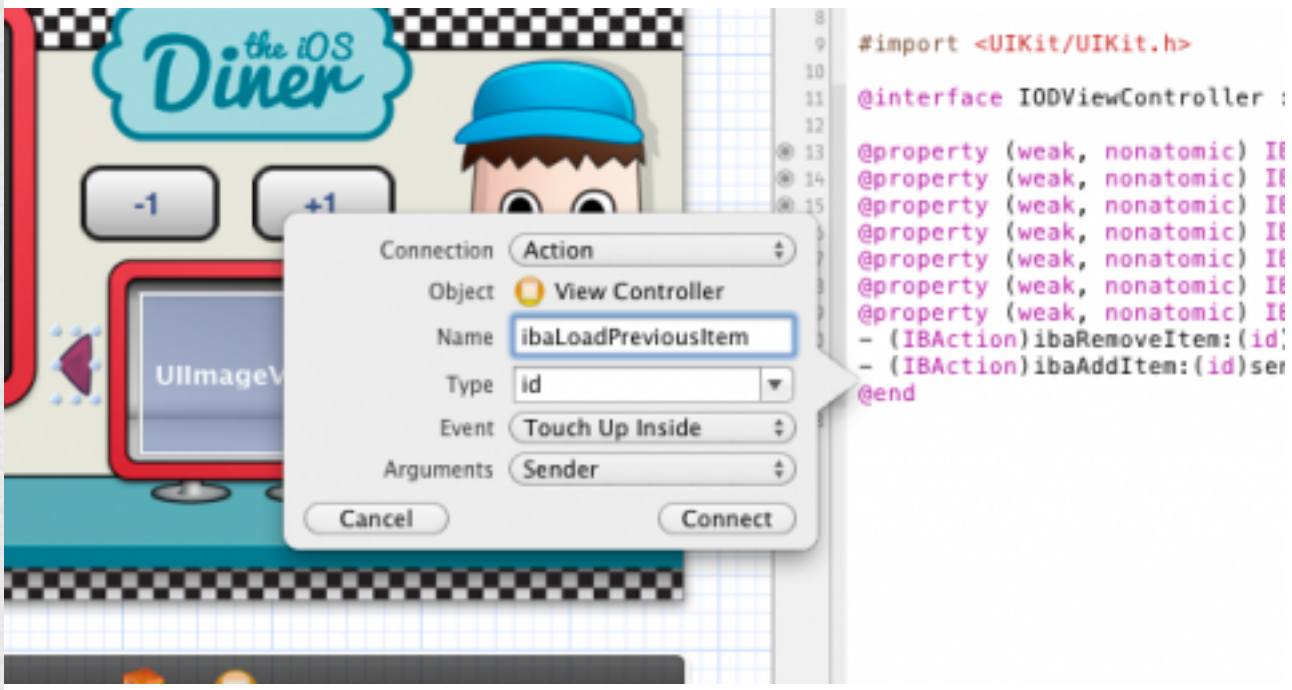
This is an IBAction, so I prefix mine with "iba." If you want to follow this convention, name this outlet "ibaRemoveItem." Then click Connect."



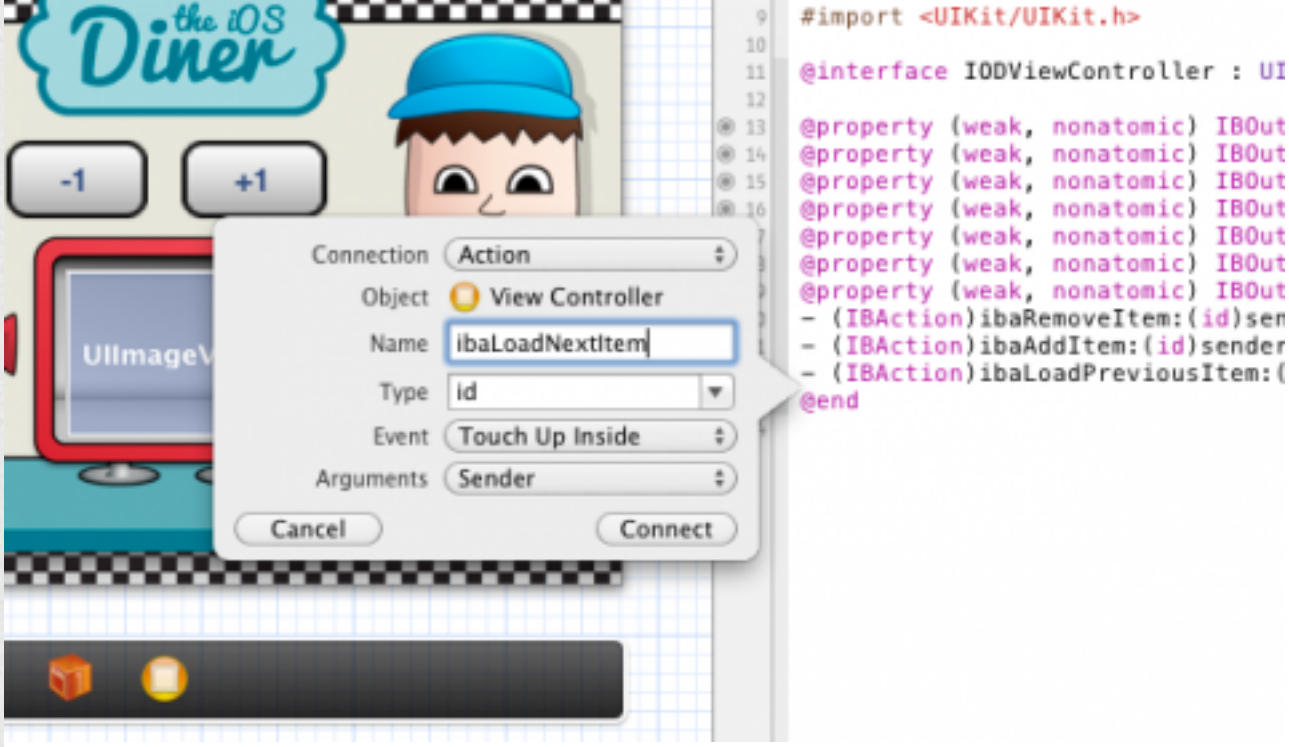
Do the same for the “+1” button, naming the action “ibaAddItem”…



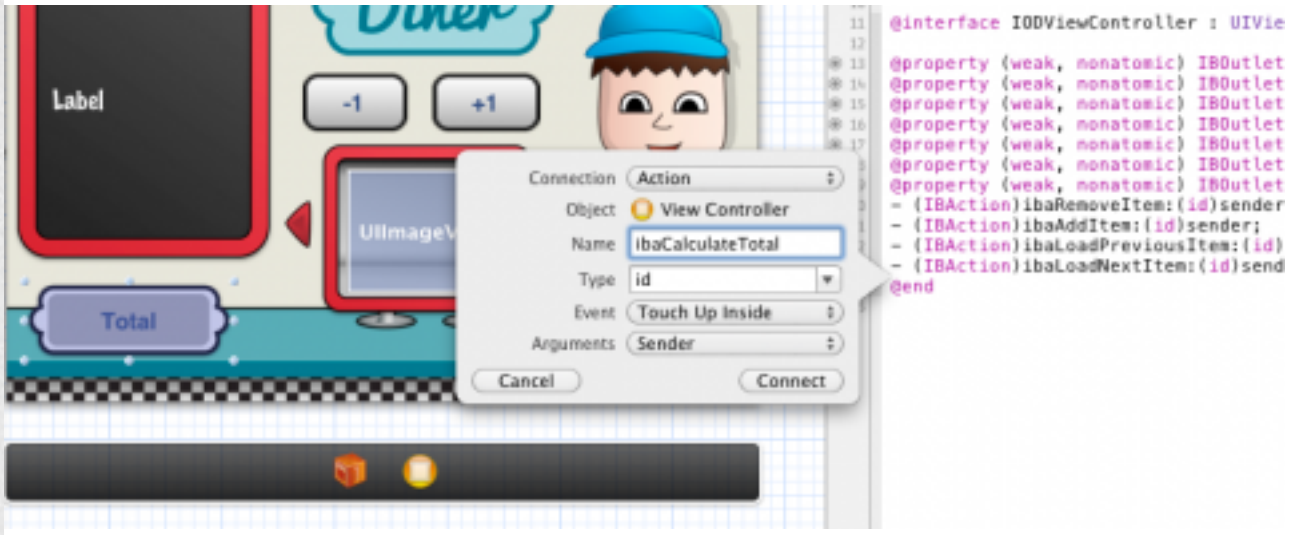
…and for the Red Left Triangle button, naming the action “ibaLoadPreviousItem”…



…and don’t forget the Red Right Triangle button. Call the action “ibaLoadNextItem”:



Finally, add an IBAction for the “Total” button in the lower left, and name it “ibaCalculateTotal.”



Setting Up Web Service

Before you get on to coding, it’s time to set up the web service. I’m not going to explain this in too much detail, as there are already several tutorials on this site that do a good job covering web services (How To Write A Simple PHP/MySQL Web Service for an iOS App and How to Write an iOS App That Uses a Web Service).

The code below shows you what the PHP code for the web service will look like:

```
<?php
function getStatusCodeMessage($status) {
    $codes = Array(
        100 => 'Continue',
        101 => 'Switching Protocols',
        200 => 'OK',
        201 => 'Created',
        202 => 'Accepted',
        203 => 'Non-Authoritative Information',
        204 => 'No Content',
        205 => 'Reset Content',
        206 => 'Partial Content',
        300 => 'Multiple Choices',
        301 => 'Moved Permanently',
        302 => 'Found',
        303 => 'See Other',
        304 => 'Not Modified',
        305 => 'Use Proxy',
        306 => '(Unused)',
        307 => 'Temporary  Redirect',
        400 => 'Bad Request',
```

```
        401 => 'Unauthorized',
        402 => 'Payment Required',
        403 => 'Forbidden',
        404 => 'Not Found',
        405 => 'Method Not Allowed',
        406 => 'Not Acceptable',
        407 => 'Proxy Authentication Required',
        408 => 'Request Timeout',
        409 => 'Conflict',
        410 => 'Gone',
        411 => 'Length Required',
        412 => 'Precondition Failed',
        413 => 'Request Entity Too Large',
        414 => 'Request-URI Too Long',
        415 => 'Unsupported Media Type',
        416 => 'Requested  Range Not Satisfiable',
        417 => 'Expectation Failed',
        500 => 'Internal  Server Error',
        501 => 'Not Implemented',
        502 => 'Bad Gateway',
        503 => 'Service Unavailable',
        504 => 'Gateway Timeout',
        505 => 'HTTP Version Not Supported'
    );

    return (isset($codes[$status])) ? $codes[$status] : '';
}

 // Helper method to send a HTTP response code/message
function sendResponse($status = 200, $body = '', $content_type = 'text/html') {
    $status_header = 'HTTP/1.1 ' . $status . '     ' . getStatusCodeMessage($status);
    header($status_header);
    header('Content-type:     ' . $content_type);
    echo $body;
}

class InventoryAPI {
    function getInventory() {
        $inventory = array(
            array("Name"=>"Hamburger","Price"=>0.99,"Image"=>"food_hamburger.png"),

array("Name"=>"Cheeseburger","Price"=>1.20,"Image"=>"food_cheeseburger.png"),
            array("Name"=>"Fries","Price"=>0.69,"Image"=>"food_fries.png"),
            array("Name"=>"Onion Rings","Price"=>0.69,"Image"=>"food_onion-rings.png"),
            array("Name"=>"Soda","Price"=>0.75,"Image"=>"food_soda.png"),
            array("Name"=>"Shake","Price"=>1.20,"Image"=>"food_milkshake.png")
        );

        sendResponse(200, json_encode($inventory));
    }
}

sleep(5);

$api = new InventoryAPI;
$api->getInventory();
?>
```

My web service is a pretty simple PHP script that returns an array that has been JSON encoded. The array contains what's known as associative arrays under PHP (they're called dictionaries under Objective-C), that contain the name, price, and image file name for the item.

The only other thing of note in the above code is the sleep(5) statement 3 lines up from the bottom. I'm using this to simulate a slow web service, so as to better show how Blocks can help perform asynchronous operations.

You can copy the above code into a file with the .php extension and host it somewhere or simply use mine at http://adamburkepile.com/inventory/.

Where to Go From Here?

An example project for this part of the tutorial can be downloaded here.

Wow, that was a lot of stuff that had nothing to do with blocks! But now that we're done setting up the view and web service, we can get down to the fun part in the next installment: coding!

So get yourself over to Part Two, where we will use blocks to make our diner app fully-functional.

Until then, join in the forum discussion below with your questions, comments and suggestions!

This is a blog post by iOS Tutorial Team member Adam Burkepile, a full-time Software Consultant and independent iOS developer. Check out his latest app Pocket No Agenda, or follow him on Twitter.

Adam Burkepile

Adam Burkepile is currently a full-time Software Consultant and independent iOS developer. If he isn't at the computer, he's probably getting punched in the face at Krav Maga. Check out his latest app – Pocket No Agenda. You can reach him for work or just to chat at Twitter, Github, his website, or email.