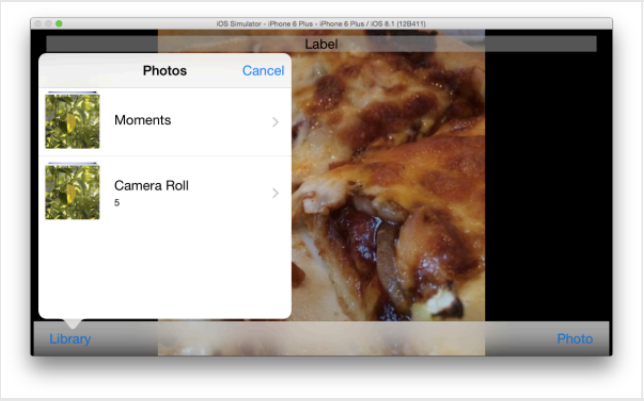
SWIFT SWIFT: USING THE UIIMAGEPICKERCONTROLLER FOR A CAMERA AND PHOTO LIBRARY.



Almost every iPhone and iPad now has a camera. Many people want to use a camera in their apps for various reasons. While you may not be building the next Instagram or Twitter, there are many places where photo documentation comes in handy.

There are two ways to use the camera. The more powerful and advanced method is using `AVFoundation`. We will discuss `AVFoundation` in a few weeks, but it is also a very complicated way to get a photo out of the camera.

The simpler way is the `UIImagePicker`. This is a super quick way as iOS 8 has simplified much of the code for it. It is also the best way to fetch photos from the photo album. In this lesson, we'll set up a basic camera and library app. In upcoming weeks we'll add a few embellishments to it that will make for a more effective camera.

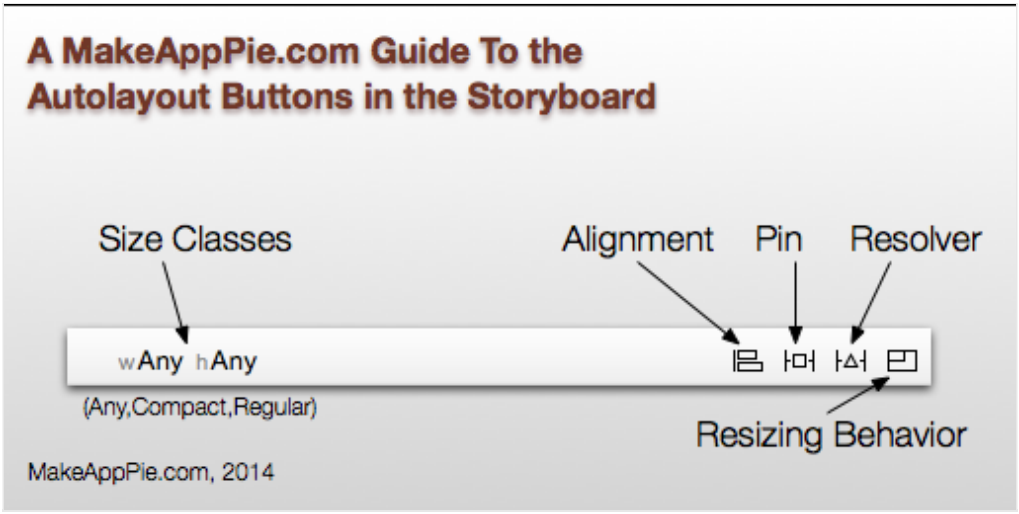
Set Up The Layout

I'm going to set this up universally. We'll be talking about some of the issues with camera on both iPad and iPhone. As a universal app we will be using auto layout. If you want a brief introduction to auto layout go over and see my layout series on YouTube. You won't need it but it may make what I do a little more understandable.

Make a new project **SwiftPizzaCam** with a single view and using **Swift**. As already mentioned, make the device **Universal**.

Go to the storyboard and check that the size classes say **w:any h:any**. Change the background color to **black**.

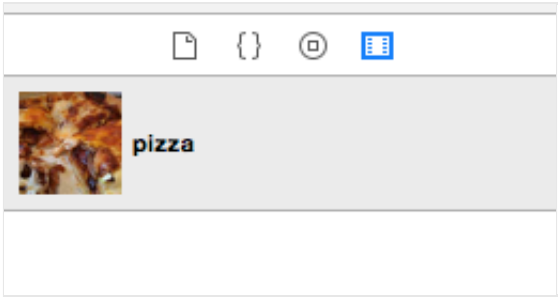
If you are not familiar with the auto layout icons, here is a guide to help you as we go through setting up the app.



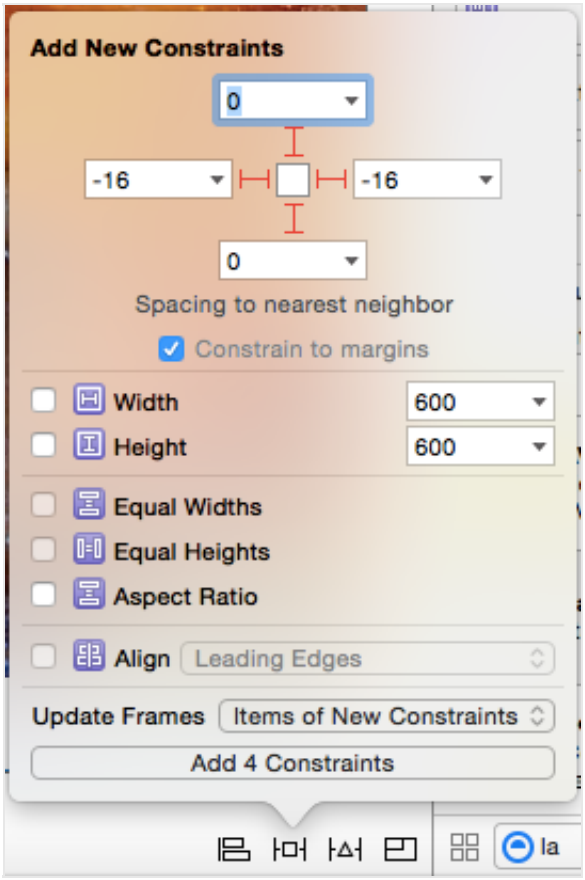
We will put a photo into an `UIImageView` when we first use the image. Right click and save the image below:

Click on **Images.Xcassets** and drag the pizza file into the assets folder. Go back to the story board

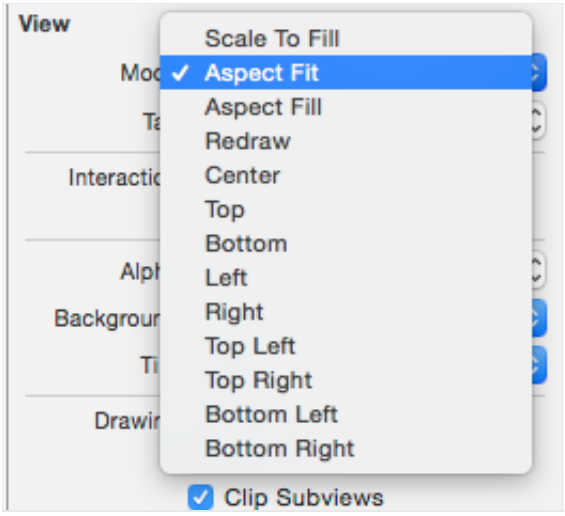
and select the clip icon. You will find the pizza media there.



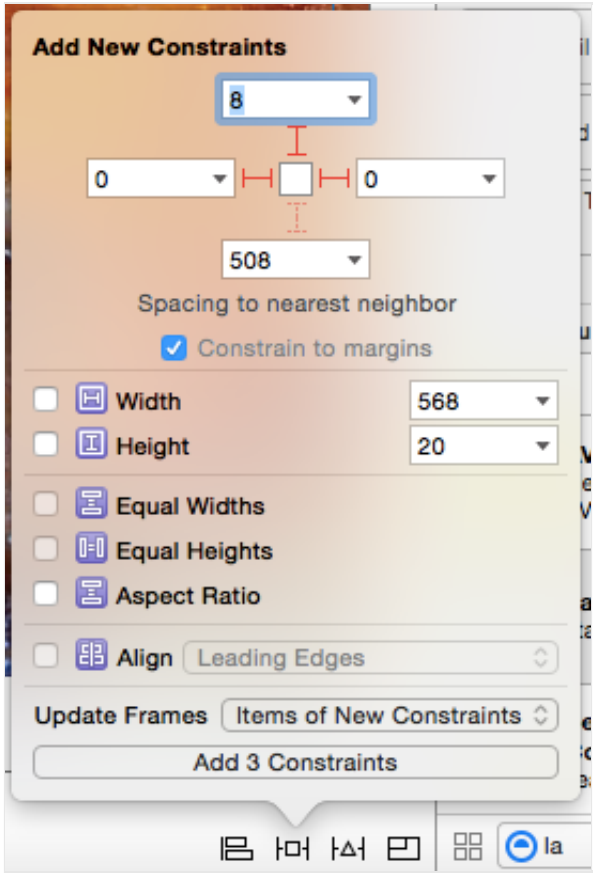
Drag out the pizza to the view and drop it into the view controller. Click the **pin** button. Click on all the i-beams, then change numbers like the photo below:



Be sure to press **tab** after you type in any number in this popover. It has an annoying habit of forgetting them if you don't. Also make sure you select **Update Frames: Items of New Constraints** towards the bottom. If you don't, you will need to click the resolver and select **Update Frame** from the upper part of the popup. In the properties change the View Mode to **AspectFit** to properly size the photo.

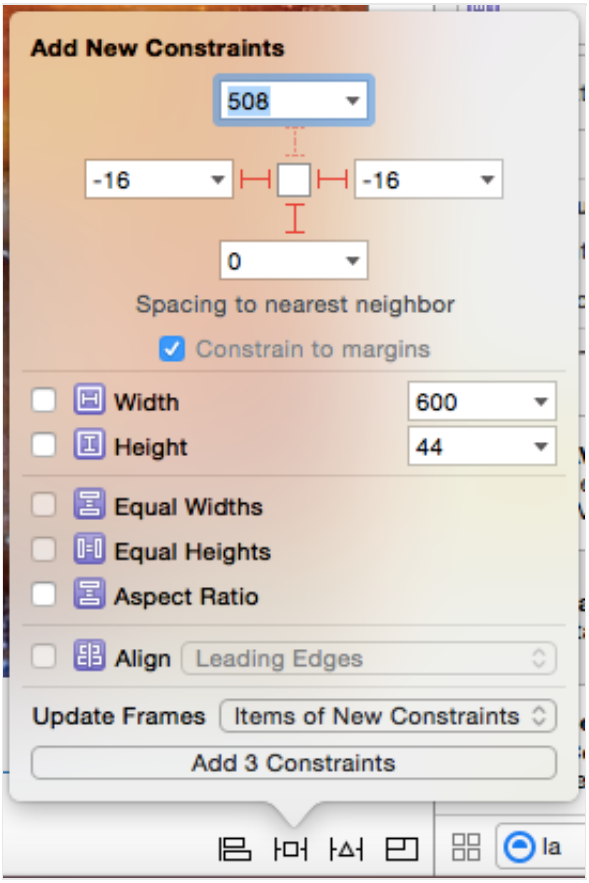


Now drag out a label onto the storyboard. Set the background property to a gray color with a 65% alpha. Center the label. Change the text of the label to read **Pizza Cam!!!**. Select the label and then pin the top left and right sides, but not the bottom, like this:

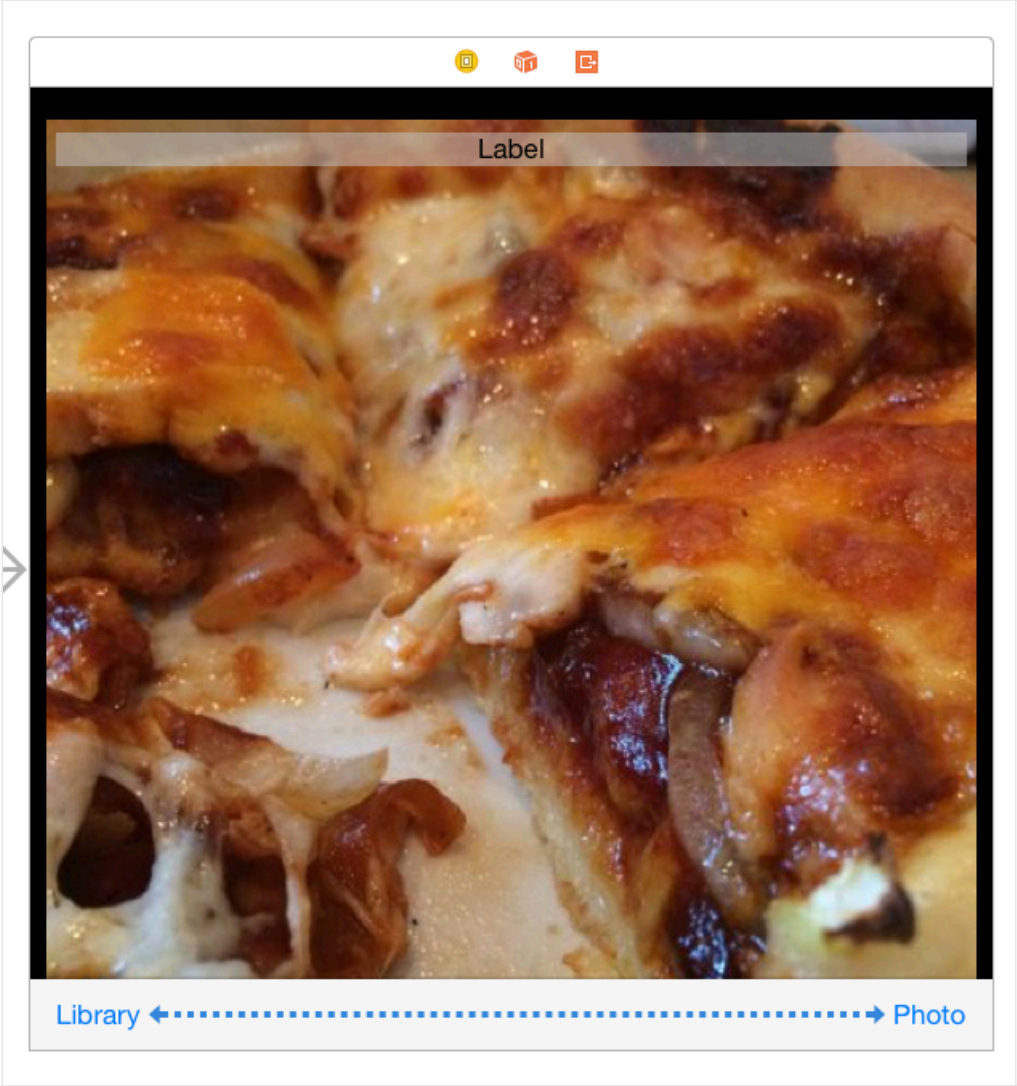


Make sure you select **Update Frames: Items of New Constraints** towards the bottom before adding the three constraints.

Drag out a toolbar and place toward the bottom of the view. Pin it to the bottom, left and right like this, again updating the frames:



Add a bar button item and a flexible space bar item to the toolbar. In one of the two bar buttons label it **Photo** and the other **Library**. Your tool bar and layout should look like this:



If you did everything correctly, you should have no auto layout warnings. If you do, go to the resolver and click **Update Frames** on the bottom half of the popover. If things get messier after doing this, clear the constraints on the bottom of the resolver, and pin everything again.

Wire Up the Outlets

Your next step is to wire up all the outlets and actions. Control-drag from the image picker and make an outlet called **myImagePicker**. Control drag from the Library button and make an action called **photoFromLibrary** with a sender of type `UIBarButtonItem`. This is important for stuff we will do later. Do this again, but with the Photo button and named shootPhoto. Again, make the sender `UIBarButtonItem`.
You can clean up the code a bit if you like. When done you should have something like this:

```
1
```

```
2
3  import UIKit
4  class ViewController: UIViewController{
5      @IBOutlet weak var myImageView: UIImageView!
6      @IBAction func shootPhoto(sender: UIBarButtonItem){
7  }
8      @IBAction func photofromLibrary(sender: UIBarButtonItem) {
9     }
10     override func viewDidLoad() {
11         super.viewDidLoad()
12         // Do any additional setup after loading the view, typically from a
   nib.
13     }
14 }
15
16
```

Add the UIImagePickerController and Delegate

The star of our show the `UIImagePickerController` is missing. we do that programmatically. Close the assistant editor, and open **ViewController.swift**. Add the following line under the outlet for `myImageView`:

```
1  let picker = UIImagePickerController()
```

The `UIImagePicker` does much of its works from a delegate. Add the following to the class description:

```
1  class ViewController:
   UIViewController,UIImagePickerControllerDelegate,UINavigationControllerDelegate
   {
```

We actually have two delegates: `UIImagePickerControllerDelegate` and `UINavigationControllerDelegate`. The `UINavigationControllerDelegate` is required but we do nothing with it. In the `viewDidLoad`, add the following:

```
override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.
        picker.delegate = self
    }
```

We have wired up the delegate. Unlike other delegates, there is no required methods. However, this will not work without implementing two methods. At the bottom of the class, add the following:

```
//MARK: Delegates
func imagePickerController(picker: UIImagePickerController,
didFinishPickingMediaWithInfo info: [NSObject : AnyObject]) {
}
func imagePickerControllerDidCancel(picker: UIImagePickerController) {
}
```

These two methods handle our selections in the library and camera. We can either handle the cancel case with `imagePickerControllerDidCancel` or handle media with `didFinishPickingMediaWithInfo`

Getting a Photo from the Library

The `UIImagePickerController` is a view controller that gets presented modally. When we select or cancel the picker, it runs the delegate, where we handle the case and dismiss the modal. Let's implement the photo library first, then the delegates. Add the following code to the `photoFromLibrary` method:

```
@IBAction func photofromLibrary(sender: UIBarButtonItem) {
picker.allowsEditing = false //2
picker.sourceType = .PhotoLibrary //3
presentViewController(picker, animated: true, completion: nil)//4
```

```
5 }
```

To get to the photo picker, it is really only three lines of code. We already initialized `picker`. In line two, we tell the picker we want a whole picture, not an edited version. In line three we set the source type to the photo library, and line four we use `presentViewController` to present the picker in a default full screen modal.

If you build and run, you would be able to press the `photoFromLibrary` method and then get stuck in the library. We need delegates to get out of the library. First let's add the code for the cancel delegate:

```
1 func imagePickerControllerDidCancel(picker: UIImagePickerController) {
2 dismissViewControllerAnimated(true, completion: nil)
3 }
```

This does nothing special: it dismisses the modal controller. If we do pick a photo, we want to do something with the photo. Add the following code to the other delegate method:

```
1 func imagePickerController(picker: UIImagePickerController,
  didFinishPickingMediaWithInfo info: [NSObject : AnyObject]) {
2 var chosenImage = info[UIImagePickerControllerOriginalImage] as UIImage //2
3 myImageView.contentMode = .ScaleAspectFit //3
4 myImageView.image = chosenImage //4
5 dismissViewControllerAnimated(true, completion: nil) //5
6 }
```

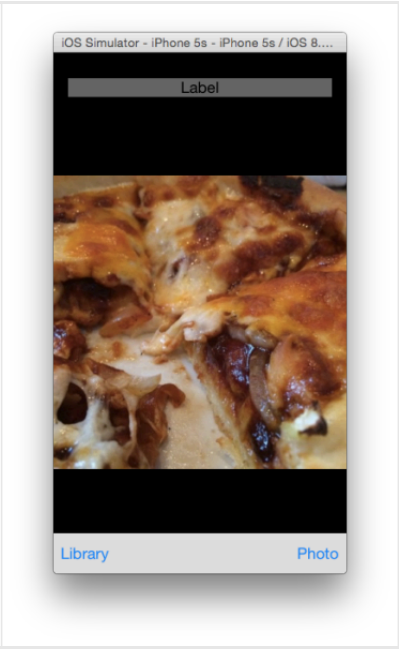
One of the parameters of this method is `info`. It has a dictionary of various information about the selected media, including metadata, a user edited image if the `.allowsEditing` property is `true`, and a `NSURL` location of the image. For our camera app, the important key is the `UIImagePickerControllerOriginalImage`. We take that key and put the image into a variable. We could make this a constant, but there is some future iterations of this project that will need this as a variable.

Camera images are usually bigger than a `UIImageView` on a device. To shrink them quickly to a visible size, the best way is to set the `.contentView` property of our `UIImageView` to

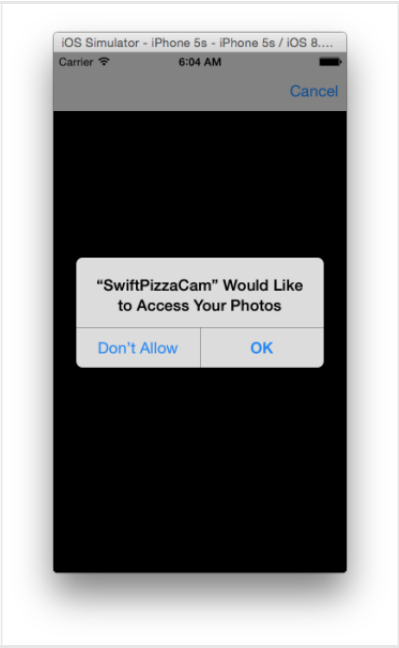
`.ScaleAspectFit` as we do in line 3.

Running as an iPhone and iPad app: Using Popovers

Run in the simulator as a iPhone 5s. You should get a screen like this:

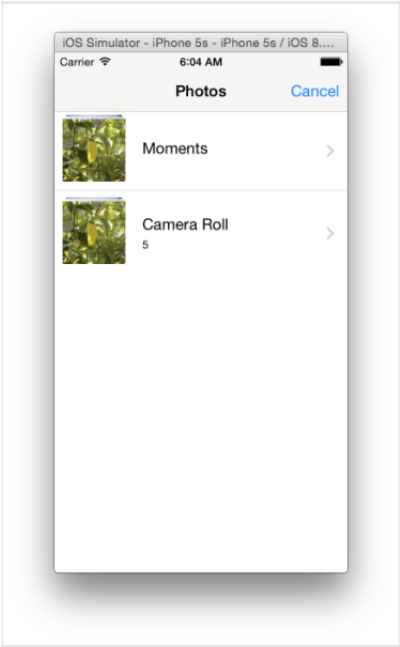


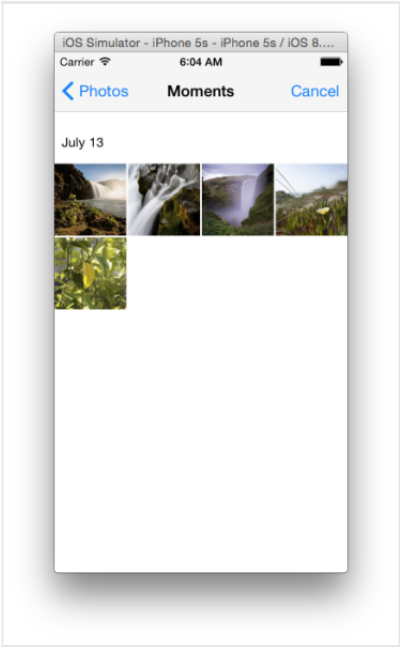
Tap the library bar button. You will get an alert:



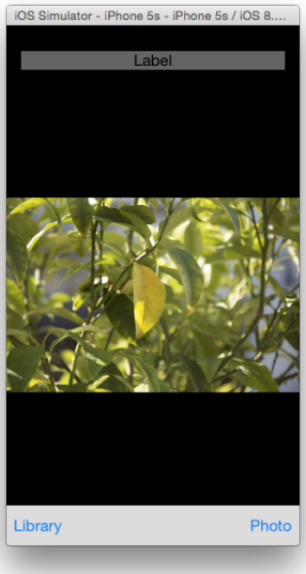
Apple requires any use of the camera, the photo library or any personal information for that matter

asks the user to agree sharing information with the app on the first use. For the `UIImagePicker`, this is included in your first run of the app. Tap **OK**. We get our Image picker,

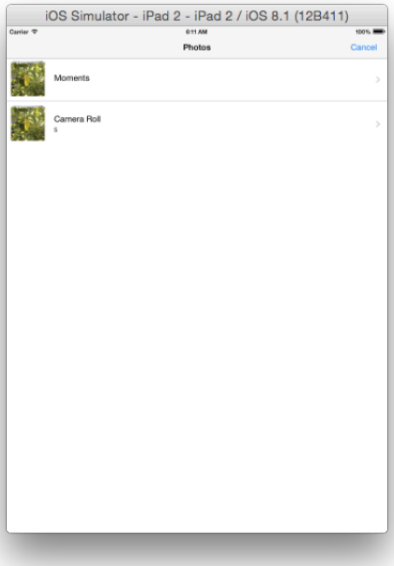




When we tap a photo, it appears on the app.



Stop the app and run as an iPad 2. After answering **OK** to the library privacy question you get this:



Which looks fine, *except* Apple doesn't like it. They want you to use a popover. If you present this as a modal using `presentViewController` in anything before iOS 8 , you will get an exception and crash the app — Apple forces you to use a popover. All that changed in iOS 8 when modals and popovers merged into `presentViewController`. The class documentation still notes that popovers are required for accessing the photo library on iPads when using the newly deprecated way of presenting a popover. I cannot find the correct way in the *Human Interface Guide*, but I

think it is still good style to use a popover. Fortunately in iOS 8, it is very easy to get the popover to work. Change this in `photoFromLibrary`

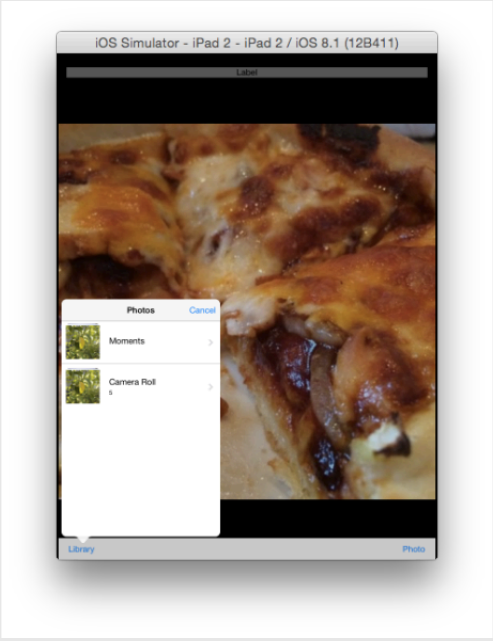
```
1 presentViewController(picker, animated: true, completion: nil)//4
```

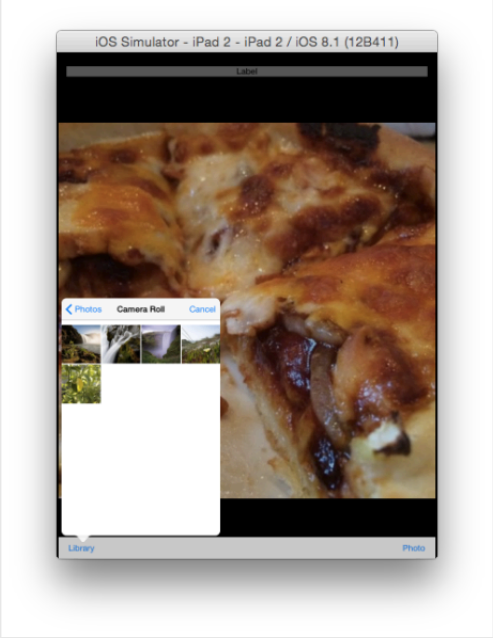
to this:

```
1 picker.modalPresentationStyle = .Popover
2 presentViewController(picker, animated: true, completion: nil)//4
3 picker.popoverPresentationController?.barButtonItem = sender
```

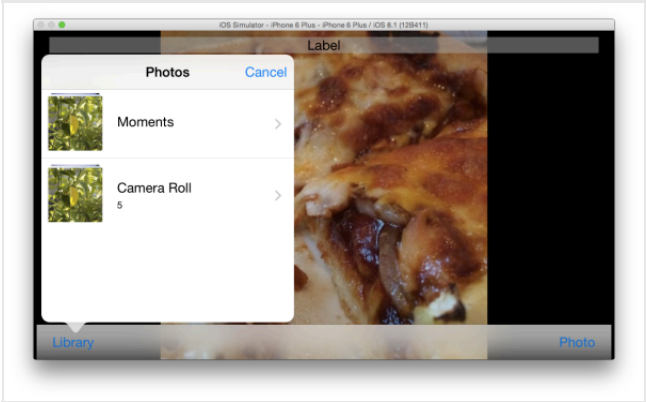
Line 1 selects a presentation style of a popover. We then present the popover, and set the reference point for the popover to the bar button item. As I mentioned in another post popovers need a reference rectangle to pop up from. In this case, we can use the `UIBarButtonItem` which happens to be sender . This is why it was so important to use `UIBarButtonItem` as the sender type.

Build and run with the iPad 2 simulator. Now we have a popover to pick a photo.





I don't know if using a full Screen modal will cause an app to be rejected for iOS 8, but since it is not hard to add, and `presentViewController` will present the right controller for the space available, this is probably a good idea — especially since it solves lots of problems with the iPhone 6 plus, which uses both popovers and modals, depending on orientation:



We save ourselves from hours of device fragmentation work this way.

Adding a Camera: Where Simulators Fear to Tread.

Basic Camera code is almost the same as adding a photo library. Change the `shootPhoto` code to this:

```
1 @IBAction func shootPhoto(sender: UIBarButtonItem) {
2 picker.allowsEditing = false
```

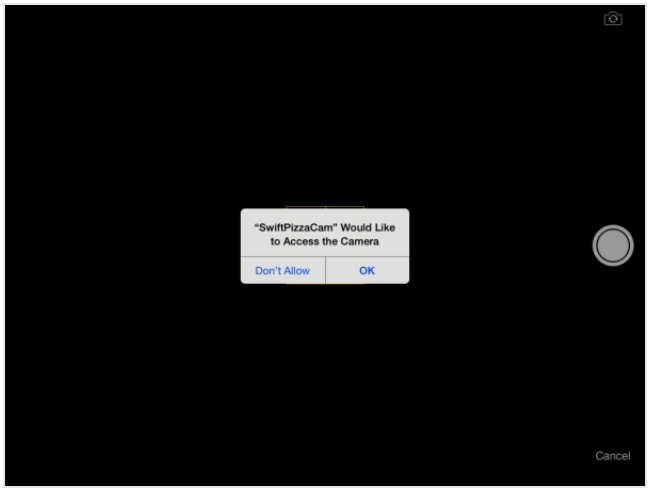
```
3 picker.sourceType = UIImagePickerControllerSourceType.Camera
4 picker.cameraCaptureMode = .Photo
5 presentViewController(picker, animated: true, completion: nil)
6 }
```

We changed the `sourceType` property to `.Camera` and specified the `cameraCaptureMode` property to `.Photo`. Camera, unlike the photo library, is best full screen, so we don't make it a popover.

If you build and run this in the simulator, you will crash.
`SwiftPizzaCam[1649:42600] *** Terminating app due to uncaught exception 'NSInvalidArgumentException', reason: 'Source type 1 not available'`

In order to test and use camera code you *have* to use a real connected device. This will require an Apple developer's license and setting up the permission for the device. I connected my iPad mini and ran the code:

When I pressed the photo button, I again get the message about allowing access.



tapping OK, I get the camera:

Take a picture, and the app asks me if I want to keep it.

and the app returns with my photo.

This all works, but we really want to prevent that crash. Since we are using iOS 8, we dont have to

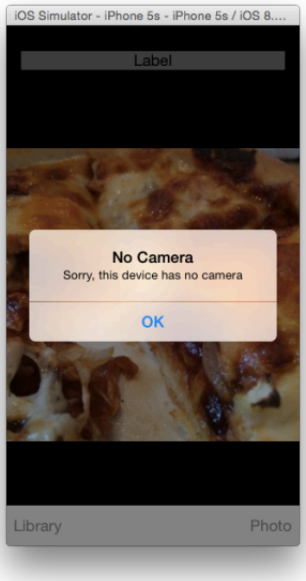
code so much as earlier versions due to a truth about our hardware: All the hardware that iOS 8 works with has a rear camera. We just have to look for a rear camera. One way to do that is with a `UIImagePickerController` class method `availableCaptureModesForCameraDevice`. If `nil`, there is no hardware. Change the method to this:

```
1  @IBAction func shootPhoto(sender: UIBarButtonItem) {
2      if UIImagePickerController.availableCaptureModesForCameraDevice(.Rear) != nil {
3          picker.allowsEditing = false
4          picker.sourceType = UIImagePickerControllerSourceType.Camera
5          picker.cameraCaptureMode = .Photo
6          presentViewController(picker, animated: true, completion: nil)
7          } else {
8              noCamera()
9          }
10 }
```

Line 2 checks if there are any devices. If there is, run the camera. If not, execute our handler code `noCamera`. For that handler we'll add an alert like this, using the new iOS 8 `UIAlertController`:

```
1  func noCamera(){
2      let alertVC = UIAlertController(title: "No Camera", message: "Sorry, this device has no camera", preferredStyle: .Alert)
3      let okAction = UIAlertAction(title: "OK", style:.Default, handler: nil)
4      alertVC.addAction(okAction)
5      presentViewController(alertVC, animated: true, completion: nil)
6  }
```

Line 2 makes an alert with the proper message. We add an **OK** action button in lines 3 and 4, then present the alert as a modal in line 5. With this code, if you run in the simulator, you get this when you attempt to take a picture:



The Basics, but Wait! There's more!

Basic `UIImagePickerController` is relatively easy to implement, but there are a lot of issues it leaves hanging:

- Hitting no for privacy settings for accessing the camera or library
- Dealing with more than one popover
- Customizing and adding more controls for the camera
- Adding and playing video
- Using and storing pictures
- Using a UIScrollView to zoom and scroll around a picture.
- Getting rid of the memory warnings
- Wanting more power over my camera controls
- Sharing pictures with my friends

Many of these could be questions that can be answered in context with the camera app, or outside of that context. In the next few weeks I'll be answering them both ways. We'll have posts about individual concepts, like a post on `UIScrollView` and then a post on how to add it to the camera app.

The Whole Code

```
1  //
2  //  ViewController.swift
3  //  SwiftPizzaCam
4  //
5  //  Created by Steven Lipton on 12/3/14.
6  //  Copyright (c) 2014 Steven Lipton. All rights reserved.
7  //
8  // Basic a camera app that takes pictures and grabs them for a background from
   the photo library
9
   import UIKit
10
   class ViewController:
11 UIViewController,UIImagePickerControllerDelegate,UINavigationControllerDelegate
   {
12
       @IBOutlet weak var myImageView: UIImageView!
13
       let picker = UIImagePickerController()   //our controller.
14
                                                 //Memory will be conserved a bit
15 if you place this in the actions.
16                                              // I did this to make code a bit
   more streamlined
17
       //MARK: - Methods
18
   // An alert method using the new iOS 8 UIAlertController instead of the
19 deprecated UIAlertview
20 // make the alert with the preferredstyle .Alert, make necessary actions, and
   then add the actions.
21
   // add to the handler a closure if you want the action to do anything.
22
       func noCamera(){
23
           let alertVC = UIAlertController(title: "No Camera", message: "Sorry,
24 this device has no camera", preferredStyle: .Alert)
25         let okAction = UIAlertAction(title: "OK", style:.Default, handler: nil)
26
           alertVC.addAction(okAction)
27
           presentViewController(alertVC, animated: true, completion: nil)
28
       }
```

```
29      //MARK: - Actions
30  //get a photo from the library. We present as a popover on iPad, and fullscreen
    on smaller devices.
31
32      @IBAction func photoFromLibrary(sender: UIBarButtonItem) {
33
34          picker.allowsEditing = false //2
35
36          picker.sourceType = .PhotoLibrary //3
37
38          picker.modalPresentationStyle = .Popover
39
40          presentViewController(picker, animated: true, completion: nil)//4
41
42          picker.popoverPresentationController?.barButtonItem = sender
43
44      }
45  //take a picture, check if we have a camera first.
46
47      @IBAction func shootPhoto(sender: UIBarButtonItem) {
48
49          if UIImagePickerController.availableCaptureModesForCameraDevice(.Rear)
    != nil {
50          picker.allowsEditing = false
51          picker.sourceType = UIImagePickerControllerSourceType.Camera
52          picker.cameraCaptureMode = .Photo
53          presentViewController(picker, animated: true, completion: nil)
54          } else {
55              noCamera()
56          }
      }
      override func viewDidLoad() {
          super.viewDidLoad()
          // Do any additional setup after loading the view, typically from a
    nib.
          picker.delegate = self   //the required delegate to get a photo back to
    the app.
      }
      //MARK: - Delegates
```

```
57 //What to do when the picker returns with a photo
58     func imagePickerController(picker: UIImagePickerController,
   didFinishPickingMediaWithInfo info: [NSObject : AnyObject]) {
59
         var chosenImage = info[UIImagePickerControllerOriginalImage] as UIImage
60 //2
61         myImageView.contentMode = .ScaleAspectFit //3
62         myImageView.image = chosenImage //4
63         dismissViewControllerAnimated(true, completion: nil) //5
64     }
65 //What to do if the image picker cancels.
66     func imagePickerControllerDidCancel(picker: UIImagePickerController) {
67         dismissViewControllerAnimated(true, completion: nil)
68     }
69 }
```