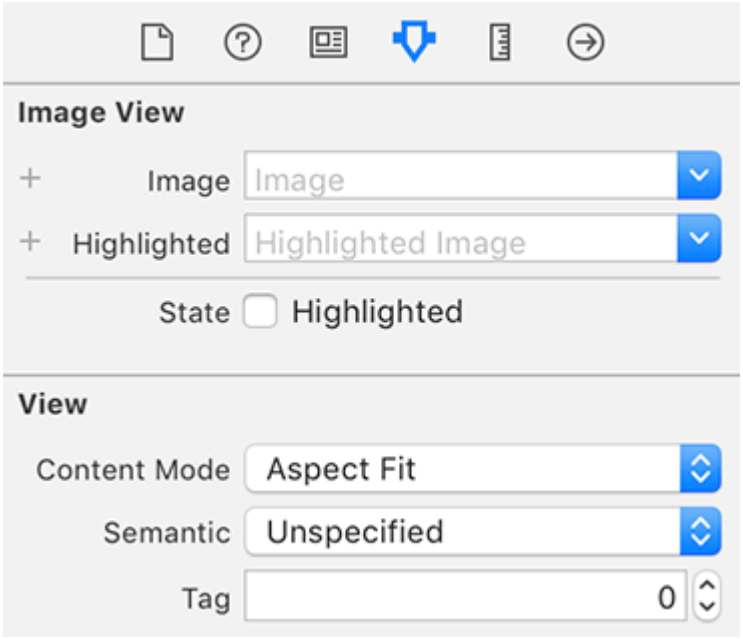
Camera-1

In this assignment you will add an image view to your Home Owner detail screen so your users can add a photo to an item. You will also integrate the user's camera so they can take a picture or select a photo from their saved photos.

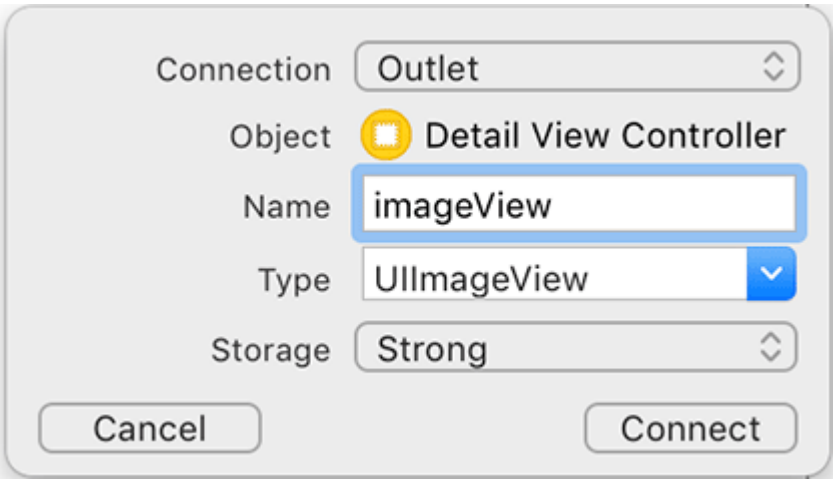
1. Open your Home Owner project and then open Main.storyboard.

2. Open Main.storyboard and drag an instance of UIImageView onto the view at the bottom of the stack view. Select the image view and open its size inspector. You want the vertical content hugging and content compression resistance properties for the image view to be lower than those of the other views. Change the vertical content hugging priority to be 248 and the vertical content compression resistance priority to be 749.

3. With the UIImageView selected, open the attributes inspector. Find the Content Mode attribute and change it to Aspect Fit. You will not see a change on the storyboard, but now images will be resized to fit within the bounds of the UIImageView:

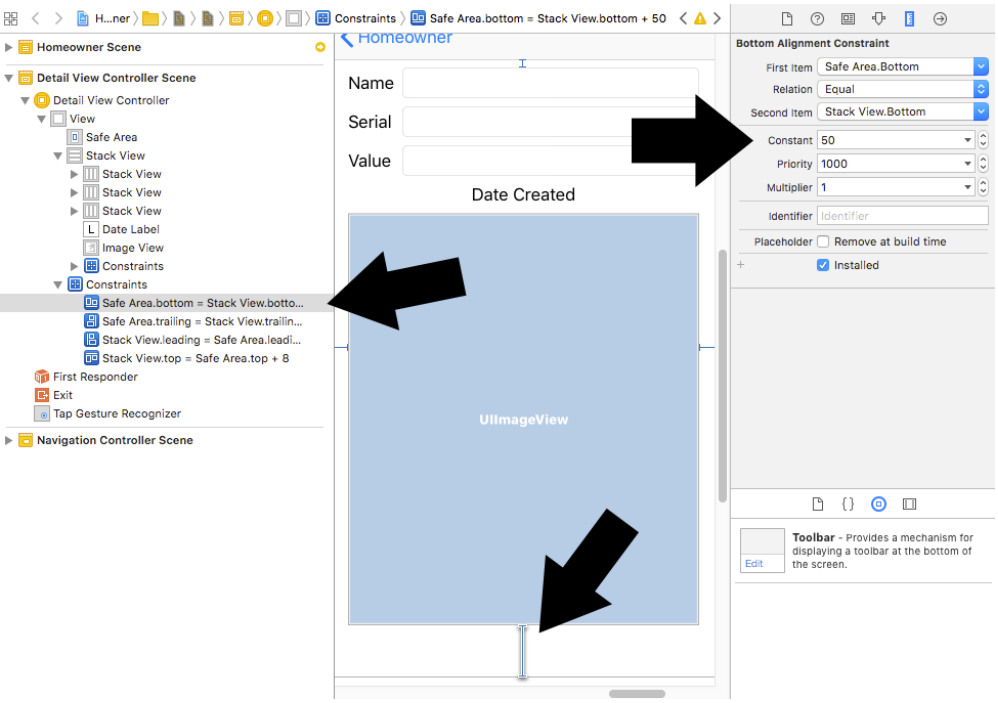


4. Next, Open DetailViewController.swift side-by-side the storyboard by clicking the two circles in the top right to open the assistant. Control-drag from the UIImageView to the top of the DetailViewController class. Name the outlet imageView and make sure the storage type is Strong. Click Connect:



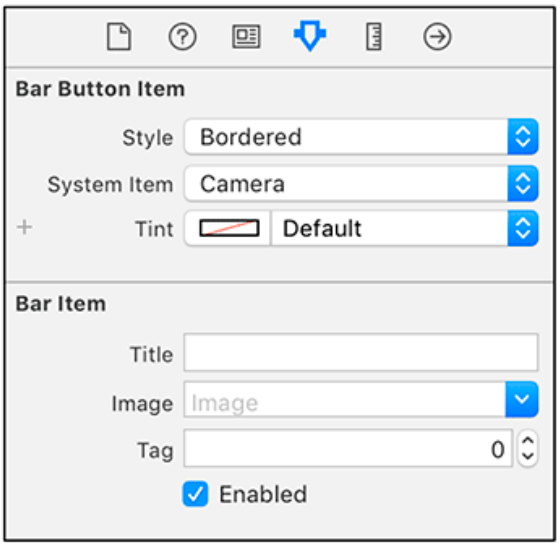
5. Close the assistant editor to give yourself more room to work in the storyboard.

6. We now need to make room for our toolbar that we will put at the bottom of our screen. Open your document outline and find the constraint that sets the space from the bottom of your image view to the bottom of the screen (a.k.a. Safe Area) as seen below. Once you have the constraint selected, open its size inspector and change the constant to 50.



7. Now drag a toolbar from the object library onto the bottom of the detail view controller.

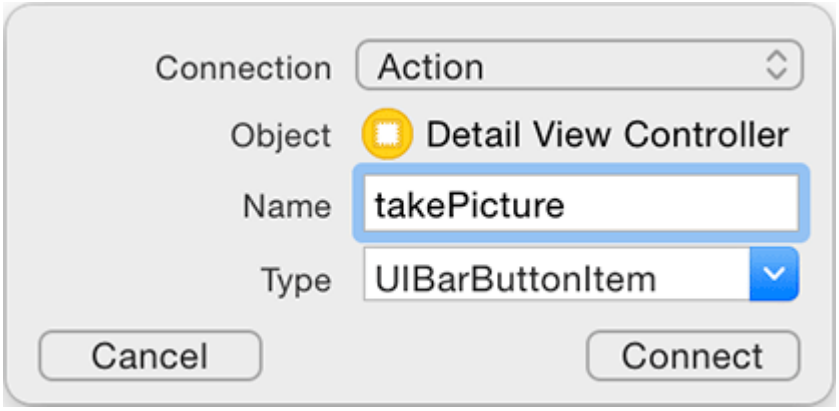
8. By default, a new instance of UIToolbar that is created in an interface file comes with one UIBarButtonItem. Select this bar button item and open the attributes inspector. Change the System Item to Camera, and the item will show a camera icon:



9. With Main.storyboard still open, Open DetailViewController.swift in the assistant editor.

10. In Main.storyboard, select the camera button by first clicking on the toolbar and then the button itself. Control-drag from the selected button to DetailViewController.swift.

11. In the Connection pop-up menu, select Action as the connection type, name it takePicture, select UIBarButtonItem as the type, and click Connect:



12. If you made any mistakes while making this connection, you will need to open Main.storyboard and disconnect any bad connections. (Look for yellow warning signs in the connections inspector.)

13. In DetailViewController.swift, find the stub for takePicture(_:). Add the following code to create the image picker and set its sourceType:

```
@IBAction func takePicture(_ sender: UIBarButtonItem) {
    let imagePicker = UIImagePickerController()

    if UIImagePickerController.isSourceTypeAvailable(.camera) {
        imagePicker.sourceType = .camera
    } else {
        imagePicker.sourceType = .photoLibrary
    }
}
```

14. At the top of DetailViewController.swift, declare that DetailViewController conforms to the UINavigationControllerDelegate and the UIImagePickerControllerDelegate protocols:

```
class DetailViewController: UIViewController, UITextFieldDelegeate,
UINavigationControllerDelegate, UIImagePickerControllerDelegate {
```

15. In DetailViewController.swift, set the instance of DetailViewController to be the image picker's delegate in takePicture(_:):

```
@IBAction func takePicture(_ sender: UIBarButtonItem) {
    let imagePicker = UIImagePickerController()

    if UIImagePickerController.isSourceTypeAvailable(.camera) {
        imagePicker.sourceType = .camera
    } else {
        imagePicker.sourceType = .photoLibrary
    }

    imagePicker.delegate = self
}
```

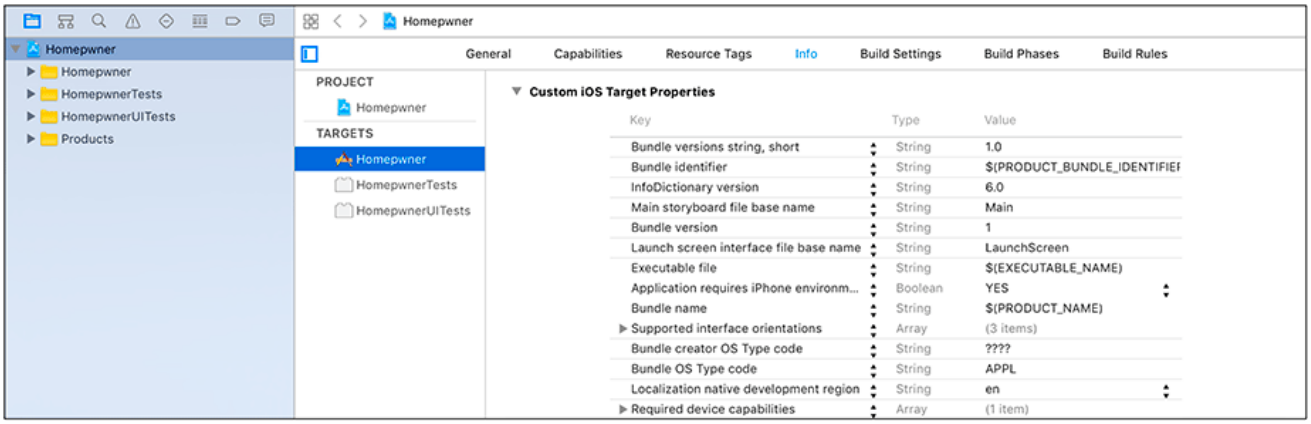
16. In DetailViewController.swift, add code to the end of takePicture(_:) to present the UIImagePickerController:

```
present(imagePicker, animated: true, completion: nil)
```

17. Build and run the application. Select an Item to open the detail view and then tap the camera button on the UIToolbar and … the application crashes. Take a look at the description of the crash in the console:

    Homepwner[3575:64615] [access] This app has crashed because it attempted to access privacy-sensitive data without a usage description. The app's Info.plist must contain an NSPhotoLibraryUsageDescription key with a string value explaining

18. In the project navigator, select the project at the top. Make sure the Homepwner target is selected and open the Info tab along the top:

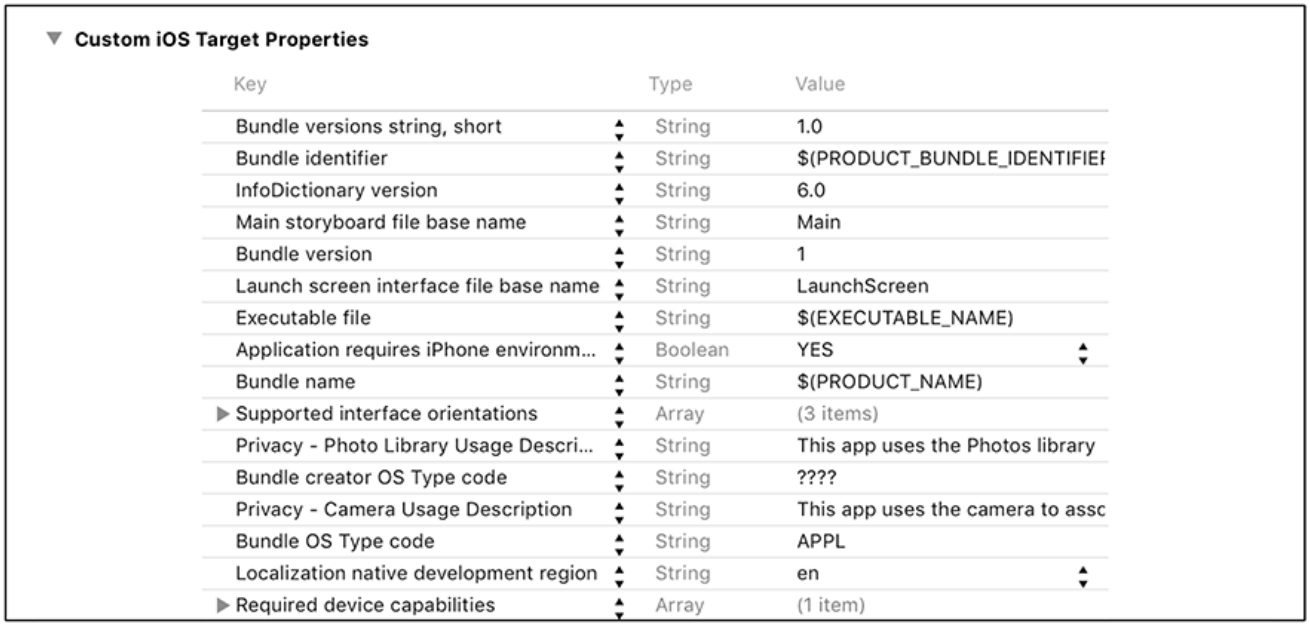


19. Hover over the last entry in this list of Custom iOS Target Properties and click the + button. Set the Key of this new entry to be NSCameraUsageDescription and the Type to be a String.

20. Double-click on the Value for this new row and enter the string “This app uses the camera to associate photos with items.” This is the string that will be presented to the user.

21. Now repeat the same steps above to add a usage description for the photo library. The Key will be NSPhotoLibraryUsageDescription of type String and the Value will be “This app uses the Photos library to associate photos with items.”

22. Your Custom iOS Target Properties should look something like this (they may be in a different order):



23. In DetailViewController.swift, implement the imagePickerController(_:didFinishPickingMediaWithInfo:) method to put the image into the UIImageView and then call the method to dismiss the image picker:

```
func imagePickerController(_ picker: UIImagePickerController,
      didFinishPickingMediaWithInfo info: [String: Any]) {
   // Get picked image from info dictionary
   let image = info[UIImagePickerControllerOriginalImage] as! UIImage
   imageView.image = image
   dismiss(animated: true, completion: nil)
}
```

24. Run your app and ensure it is working properly.

25. Next we will store images to disk and only fetch them into memory when they are needed. This fetching will be done by a new class, ImageStore. When the application receives a low-memory notification, the ImageStore's cache will be flushed to free the memory that the fetched images were occupying. Create a new Swift file named ImageStore. In ImageStore.swift, define the ImageStore class and add a property that is an instance of NSCache.

```
~~import Foundation~~
import UIKit

class ImageStore {
   let cache = NSCache<NSString,UIImage>()
}
```

26. Now implement three methods for adding, retrieving, and deleting an image from the dictionary.

```
class ImageStore {
    let cache = NSCache<NSString,UIImage>()

    func setImage(_ image: UIImage, forKey key: String) {
        cache.setObject(image, forKey: key as NSString)
    }

    func image(forKey key: String) -> UIImage? {
        return cache.object(forKey: key as NSString)
    }

    func deleteImage(forKey key: String) {
        cache.removeObject(forKey: key as NSString)
    }
}
```

27. The DetailViewController needs an instance of ImageStore to fetch and store images. You will inject this dependency into the DetailViewController's designated initializer, just as you did for ItemsViewController and ItemStore in Chapter 10. In DetailViewController.swift, add a property for an ImageStore.

```
var item: Item! {
    didSet {
        navigationItem.title = item.name
    }
}
var imageStore: ImageStore!
```

28. Now do the same in ItemsViewController.swift.

```
var itemStore: ItemStore!
var imageStore: ImageStore!
```

29. Next, still in ItemsViewController.swift, update prepare(for:sender:) to set the imageStore property on DetailViewController.

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    // If the triggered segue is the "showItem" segue"
    switch segue.identifier {
    case "showItem"?:
        // Figure out which row was just tapped
        if let row = tableView.indexPathForSelectedRow?.row {

            // Get the item associated with this row and pass it along
            let item = itemStore.allItems[row]
            let detailViewController
                = segue.destination as! DetailViewController
            detailViewController.item = item
            detailViewController.imageStore = imageStore
        }
    default:
        preconditionFailure("Unexpected segue identifier.")
    }
}
```

30. Finally, update AppDelegate.swift to create and inject the ImageStore.

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions
        launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {
    // Override point for customization after application launch.
    // Create an ItemStore
    let itemStore = ItemStore()

    // Create an ImageStore
    let imageStore = ImageStore()

    // Access the ItemsViewController and set its item store and image store
    let navController = window!.rootViewController as! UINavigationController
    let itemsController = navController.topViewController as! ItemsViewController
    itemsController.itemStore = itemStore
    itemsController.imageStore = imageStore
```

31. When an image is added to the store, it will be put into the cache under a unique key, and the associated Item object will be given that key. When the DetailViewController wants an image from the store, it will ask its item for the key and search the cache for the image. Add a property to Item.swift to store the key.

```
let dateCreated: Date
let itemKey: String
```

32. The image keys need to be unique for your cache to work. While there are many ways to hack together a unique string, you are going to use the Cocoa Touch mechanism for creating universally unique identifiers (UUIDs), also known as globally unique identifiers (GUIDs). Objects of type NSUUID represent a UUID and are generated using the time, a counter, and a hardware identifier, which is usually the MAC address of the Wi-Fi card. In Item.swift, generate a UUID and set it as the itemKey.

```
init(name: String, serialNumber: String?, valueInDollars: Int) {
    self.name = name
    self.valueInDollars = valueInDollars
    self.serialNumber = serialNumber
    self.dateCreated = Date()
    self.itemKey = UUID().uuidString

    super.init()
}
```

33. Then, in DetailViewController.swift, update imagePickerController(_:didFinishPickingMediaWithInfo:) to store the image in the ImageStore.

```
func imagePickerController(_ picker: UIImagePickerController,
        didFinishPickingMediaWithInfo info: [String : Any]) {

    // Get picked image from info dictionary
    let image = info[UIImagePickerControllerOriginalImage] as! UIImage

    // Store the image in the ImageStore for the item's key
    imageStore.setImage(image, forKey: item.itemKey)

    // Put that image on the screen in the image view
    imageView.image = image

    // Take image picker off the screen -
    // you must call this dismiss method
    dismiss(animated: true, completion: nil)
}
```

34. Similarly, when an item is deleted, you need to delete its image from the image store. In ItemsViewController.swift, update tableView(_:commit:forRowAt:) to remove the item's image from the image store.

```
override func tableView(_ tableView: UITableView,
                        commit editingStyle: UITableViewCellEditingStyle,
                        forRowAt indexPath: IndexPath) {
    // If the table view is asking to commit a delete command...
    if editingStyle == .delete {
        let item = itemStore.allItems[indexPath.row]

        let title = "Delete \(item.name)?"
        let message = "Are you sure you want to delete this item?"

        let ac = UIAlertController(title: title,
                                   message: message,
                                   preferredStyle: .actionSheet)

        let cancelAction = UIAlertAction(title: "Cancel",
                                         style: .cancel,
                                         handler: nil)
        ac.addAction(cancelAction)

        let deleteAction = UIAlertAction(title: "Delete", style: .destructive,
                                         handler: { (action) -> Void in
            // Remove the item from the store
            self.itemStore.removeItem(item)

            // Remove the item's image from the image store
            self.imageStore.deleteImage(forKey: item.itemKey)

            // Also remove that row from the table view with an animation
            self.tableView.deleteRows(at: [indexPath], with: .automatic)
        })
        ac.addAction(deleteAction)

        // Present the alert controller
        present(ac, animated: true, completion: nil)
    }
}
```

35. The DetailViewController's view will appear when the user taps a row in ItemsViewController and when the UIImagePickerController is dismissed. In both of these situations, the imageView should be populated with the image of the Item being displayed. Currently, it is only happening when the UIImagePickerController is dismissed. In DetailViewController.swift, make this happen in viewWillAppear(_:).

```
override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)

    nameField.text = item.name
    serialNumberField.text = item.serialNumber
    valueField.text =
        numberFormatter.string(from: NSNumber(value: item.valueInDollars))
    dateLabel.text = dateFormatter.string(from: item.dateCreated)

    // Get the item key
    let key = item.itemKey

    // If there is an associated image with the item
    // display it on the image view
    let imageToDisplay = imageStore.image(forKey: key)
    imageView.image = imageToDisplay
}
```

36. Test your app to ensure it works properly

37. When you have completed this assignment commit and push your code to GitHub, then submit your commit ID using the LMS.