# LEARNING CORE DATA FOR iOS WITH SWIFT

A Hands-On Guide to Building Core Data Applications

TIM ROADLEY

# Learning Core Data for iOS with Swift

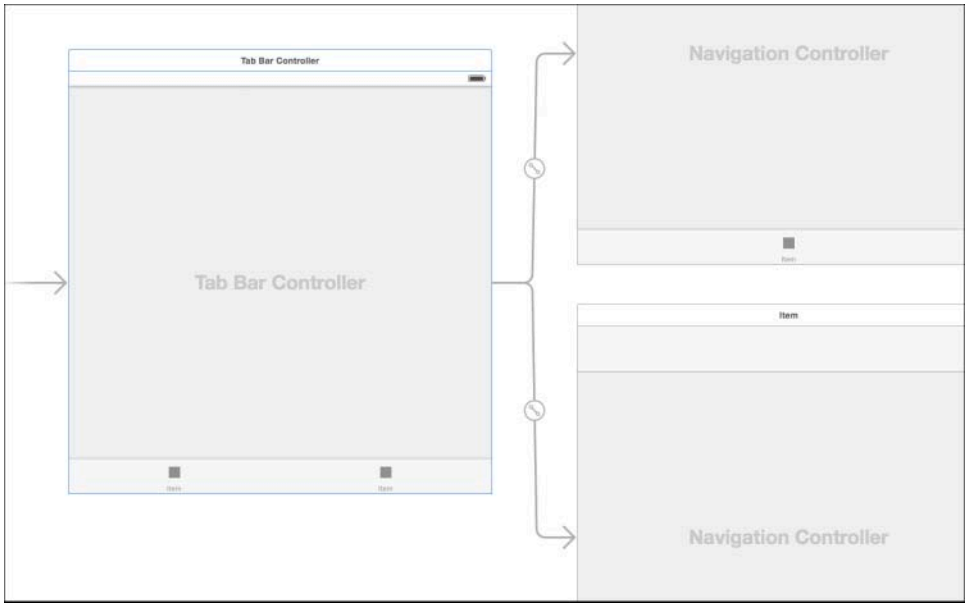


Figure 5.4 The Tab Bar Controller with two tabs

The Tab Bar Controller now has two tabs that will be used to cycle between the `PrepareTVC` and `ShopTVC` table views. Before those subclasses are prepared and assigned to the table views, final touches are needed to help identify the tabs properly.

Update Groceries as follows to add tab bar icons:

1. Download and extract the tab bar icons from the following URL: http://www.timroadley.com/LCDwS/TabBarIcons.zip.
2. Select the **Assets.xcassets** asset catalog.
3. Drag the tab bar icons into the asset catalog beneath **LaunchScreen**, as shown in Figure 5.5.

Update Groceries as follows to configure the tabs:

1. Select **Main.storyboard**.
2. Select the **Tab Bar Item** on the **Navigation Controller** next to the top table view associated with the **PrepareTVC** class.
3. Set the **Bar Item Title** to **Prepare** and **Bar Item Image** to **prepare**, as shown in Figure 5.6.

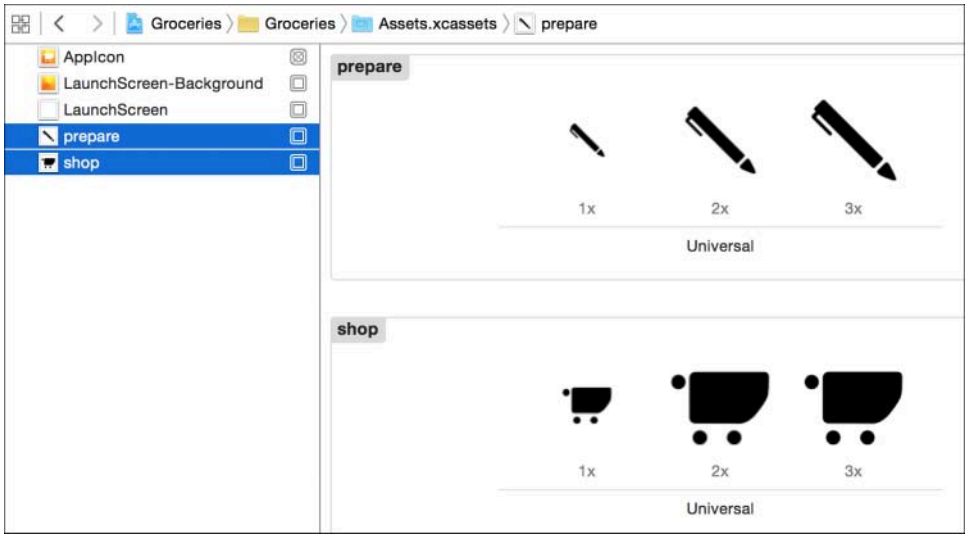


Figure 5.5 The Tab Bar Controller icons

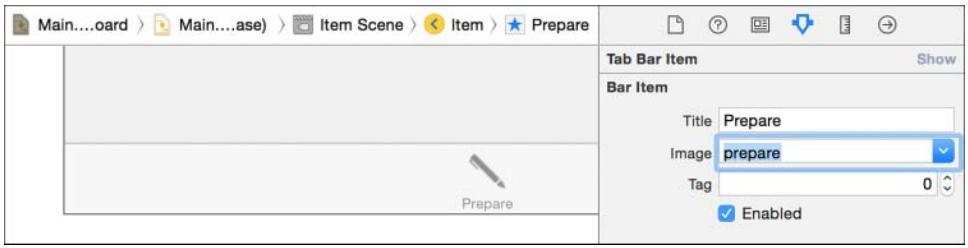


Figure 5.6 The Prepare tab

4. Select the **Tab Bar Item** on the **Navigation Controller** next to the other table view.
5. Set the **Bar Item Title** to **Shop** and **Bar Item Image** to **shop** using the technique from step 3.
6. Run the application. You should be able to switch between table views, as shown in Figure 5.7. There won't be any data in the tables just yet. If the Prepare and Shop tabs are in the wrong order, you can drag them to the correct order using the Tab Bar Controller on the storyboard.


Figure 5.7 Table view toggles in the Groceries Tab Bar Controller

## Enhancing PrepareTVC

To put an item on the shopping list (i.e., the Shop tab), a user will simply tap an item on the Prepare tab. To completely clear the shopping list, a **Clear** button will be implemented on the Prepare tab. To prevent the shopping list from being accidentally cleared, an action sheet is needed to confirm that a shopping list should in fact be cleared.

The `PrepareTVC` class now needs to be updated as a subclass of `CDTableViewController`. A valid entity and sort descriptor also need to be provided, along with some table view cell customization. Listing 5.6 shows the code involved in the updated class.

Listing 5.6 **Prepare Table View Controller (`PrepareTVC.swift`)**

```
import UIKit
import CoreData

class PrepareTVC: CDTableViewController {

    // MARK: - CELL CONFIGURATION
    override func configureCell(cell: UITableViewCell, atIndexPath
indexPath: NSIndexPath) {

        if let item = frc.objectAtIndexPath(indexPath) as? Item {

            var itemName = ""
            if let name = item.name {itemName = " \(name)"}

            // Prefix quantity and unit
            if let unit = item.unit?.name {itemName = "\(unit)\(itemName)"}
            if let quantity = item.quantity {itemName = "\(quantity)\(itemName)"}
```

```
            if let textLabel = cell.textLabel, listed = item.listed {
                textLabel.text = itemName
                cell.accessoryType = UITableViewCellAccessoryType.DetailButton

                // Color items according to the listed attribute
                if listed.boolValue {
                    textLabel.font = UIFont(name: "Helvetica Neue", size: 18)
                    textLabel.textColor = UIColor(red: 1, green: 0.2, blue: 0.2,
➥alpha: 1)
                } else {
                    textLabel.font = UIFont(name: "Helvetica Neue", size: 16)
                    textLabel.textColor = UIColor.grayColor()
                }
            } else {print("ERROR getting textLabel in \(__FUNCTION__)")}
        } else {print("ERROR getting item in \(__FUNCTION__)")}
    }

    // MARK: - INITIALIZATION
    required init?(coder aDecoder: NSCoder) {
        super.init(coder: aDecoder)

        // CDTableViewController subclass customization
        self.entity = "Item"
        self.sort = [NSSortDescriptor(key: "locationAtHome.storedIn", ascending:
➥true),NSSortDescriptor(key: "name", ascending: true)]
        self.sectionNameKeyPath = "locationAtHome.storedIn"
        self.fetchBatchSize = 50

    }

    // MARK: - VIEW
    override func viewDidAppear(animated: Bool) {
        super.viewDidAppear(animated)
        self.performFetch()
    }
}
```

The `configureCell` function sets the text of the cell's `textLabel`. The grocery item specific to the cell has the same index in the fetched results controller. This means that the `indexPath` of the cell can be used to gain reference to the appropriate managed object in `frc`. The function continues on to color-code items depending on whether they're listed on the Shop tab.

The `init` function is where entity and sort descriptor are set prior to fetching the data. This is the place you could also set any of the other optional variables from `CDTableViewController`. For example, the `self.sectionNameKeyPath` variable is set so that the shopping list is divided into sections.

The `viewDidAppear` function is used to trigger a fetch each time the view appears.

Update Groceries as follows to configure `PrepareTVC`:

1. Replace all code in `PrepareTVC.swift` with the code from Listing 5.6. If you get an error regarding optional types, make sure that the relationships in `Item+CoreDataProperties.swift` are all optional variables by suffixing them with a question mark.

## Preparing Test Data

The `PrepareTVC` is now in a position to display data, yet there's nothing in the persistent store to show because the application was deleted at the beginning of this chapter. Listing 5.7 shows the code required to add test data to the persistent store.

Listing 5.7 **Demo Data (`AppDelegate.swift demo`)**

```
func demo () {
    if CDOperation.objectCountForEntity("Item", context: CDHelper.shared.context)
➥== 0 {
        let context = CDHelper.shared.context

        let homeLocations = ["Fruit Bowl","Pantry","Nursery","Bathroom","Fridge"]
        let shopLocations = ["Produce","Aisle 1","Aisle 2","Aisle 3","Deli"]
        let unitNames = ["g","pkt","box","ml","kg"]
        let itemNames = ["Grapes","Biscuits","Nappies","Shampoo","Sausages"]

        var i = 0

        for itemName in itemNames {
            print("Inserting '\(itemName)'")
            if let locationAtHome = NSEntityDescription.insertNewObjectForEntityFor
➥Name("LocationAtHome", inManagedObjectContext: context) as? LocationAtHome
             , let locationAtShop = NSEntityDescription.insertNewObjectForEntityFor
➥Name("LocationAtShop", inManagedObjectContext: context) as? LocationAtShop
             , let unit = NSEntityDescription.insertNewObjectForEntityForName("Unit",
➥inManagedObjectContext: context) as? Unit
             , let item = NSEntityDescription.insertNewObjectForEntityForName("Item",
➥ inManagedObjectContext: context) as? Item {

                locationAtHome.storedIn = homeLocations[i]
                locationAtShop.aisle    = shopLocations[i]
                unit.name = unitNames[i]
                item.name = itemNames[i]
                item.locationAtHome = locationAtHome
                item.locationAtShop = locationAtShop
                item.unit = unit
```

```
                i++
            } else {"ERROR preparing items in \(__FUNCTION__)"}
        }
        print("Test data was inserted")
        CDHelper.saveSharedContext()
    }
}
```

Update Groceries as follows to insert some test data:

1. Replace the `demo` function of `AppDelegate.swift` with the code shown in Listing 5.7. This code inserts test data using techniques discussed in the previous chapters.
2. Add `demo()` to the `applicationDidBecomeActive` function of `AppDelegate.swift`.
3. Run the application to insert the test data, which works only if there's no existing data. The expected result is shown in Figure 5.8.

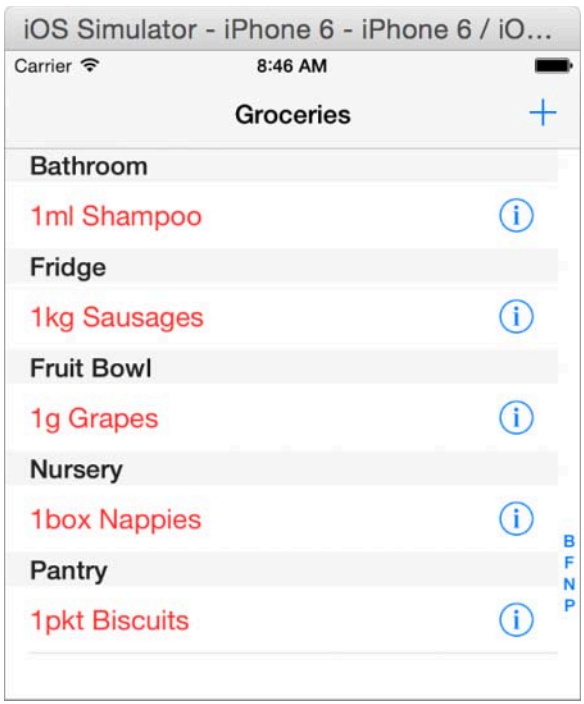


Figure 5.8 `PrepareTVC` table view with test data