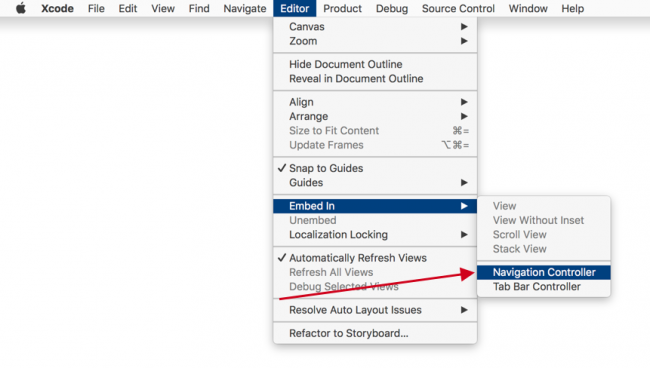
**>> TableView Example without the use of Core Data.**

**Start a new Xcode project and do the following:**

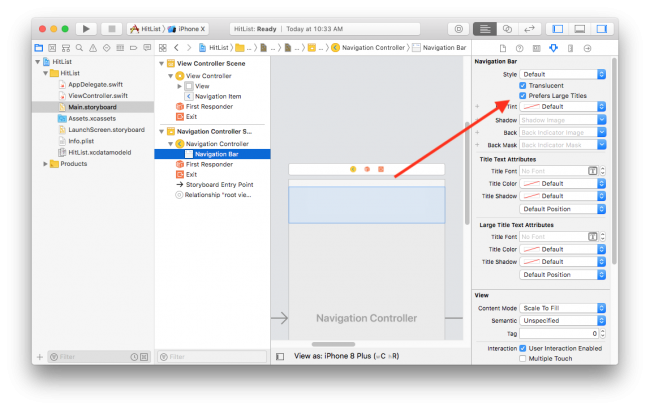
* Checking the **Use Core Data** box will cause Xcode to generate boilerplate code for what’s known as an NSPersistentContainer in **AppDelegate.swift**.
* The NSPersistentContainer consists of a set of objects that facilitate saving and retrieving information from Core Data.
* The standard stack works well for most apps, but depending on your your app and its data requirements, you can customize the stack to be more efficient.
* The idea for this sample app is simple: There will be a table view with a list of names for your very own “Employee list”.

**You’ll be able to add names to this list, and eventually, use Core Data to make sure the data is stored between sessions.**

Click on **Main.storyboard** to open it in Interface Builder. Select the view controller on the canvas and embed it inside a navigation controller. From Xcode’s **Editor** menu, select **Embed In… ▸ Navigation Controller**.



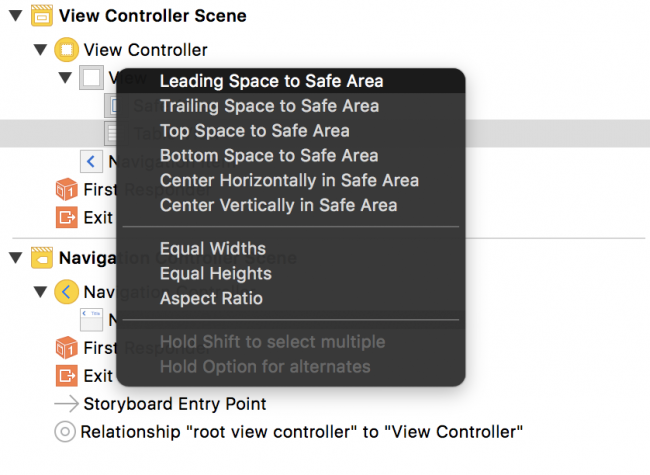
Click on the navigation controller’s navigation bar to select it, then click on **Prefers Large Titles** in the Attributes Inspector. This will give the sample app a title style that matches Apple’s stock apps.



Next, drag a **Table View** from the object library into the view controller, then resize it so it covers the entire view.

If not already open, use the icon located in the lower left corner of your canvas to open Interface Builder’s document outline.

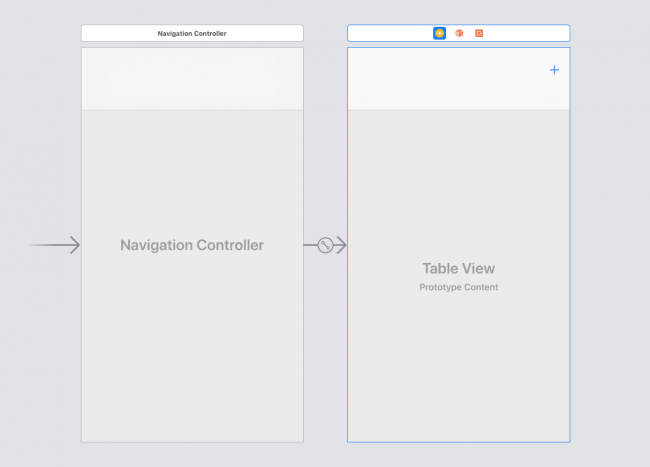
Ctrl-drag from the **Table View** in the document outline to its parent view and select the **Leading Space to Safe Area** constraint:



Do this three more times, selecting the constraints **Trailing Space to Safe Area**, **Top Space to Safe Area** and finally, **Bottom Space to Safe Area**. Adding those four constraints will make the table view fill its parent view.

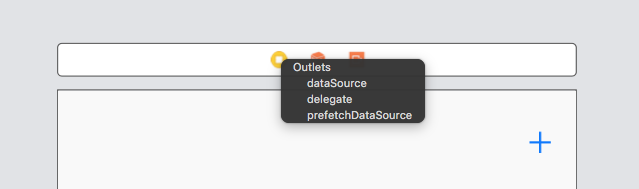
Next, drag a **Bar Button Item** and place it on the view controller’s navigation bar. Finally, select the bar button item and change its system item to **Add**.

Your canvas should look similar to the following screenshot:



Every time you tap the **Add** button, an alert controller containing a text field will appear. From there, you’ll be able to type someone’s name into the text field. Tapping Save will save the name, dismiss the alert controller and refresh the table view, displaying all the names you’ve entered.

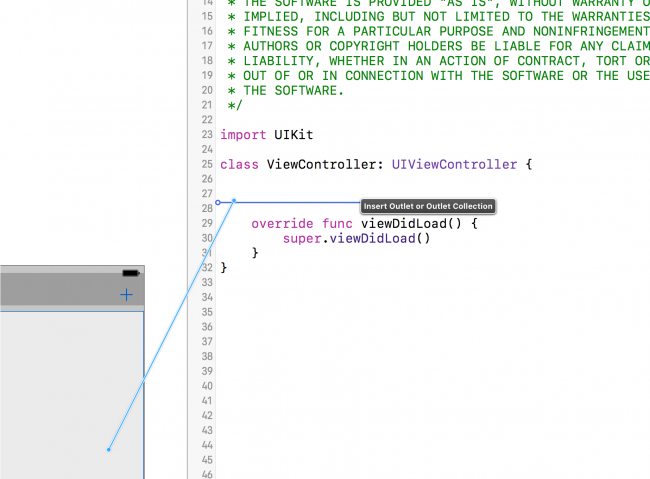
But first, you need to make the view controller the table view’s data source. In the canvas, Ctrl-drag from the table view to the yellow view controller icon above the navigation bar, as shown below, and click on **dataSource**:



In case you’re wondering, you don’t need to set up the table view’s delegate since tapping on the cells won’t trigger any action. It doesn’t get simpler than this!

Open the assistant editor by pressing Command-Option-Enter or by selecting the middle button on the Editor toolset on the Xcode bar.

Ctrl-drag from the table view onto **ViewController.swift** inside the class definition to create an IBOutlet.



Next, name the new IBOutlet property tableView, resulting in the following line:

@IBOutlet weak var tableView: UITableView!

Next, Ctrl-drag from the **Add** button into **ViewController.swift** just below your viewDidLoad() definition. This time, create an action instead of an outlet, naming the method addName, with a type UIBarButtonItem:

@IBAction func addName(\_ sender: UIBarButtonItem) {

}

You can now refer to the table view and the bar button item’s action in code.

Next, you’ll set up the model for the table view. Add the following property to **ViewController.swift** below the tableView IBOutlet:

var names: [String] = [ ]

names is a mutable array holding string values displayed by the table view. Next, replace the implementation of viewDidLoad() with the following:

override func viewDidLoad() {

super.viewDidLoad()

title = "The List"

tableView.register( UITableViewCell.self, forCellReuseIdentifier: "Cell")

} }

This will set a title on the navigation bar and register the UITableViewCell class with the table view.

**Note**: register(\_:forCellReuseIdentifier:) guarantees your table view will return a cell of the correct type when the **Cell** reuseIdentifier is provided to the dequeue method.

Next, still in **ViewController.swift**, add the following UITableViewDataSource extension below your class definition for ViewController:

}

// MARK: - UITableViewDataSource

extension ViewController: UITableViewDataSource {

func **tableView**(\_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {

return names.count

}

func **tableView**(\_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {

let cell = tableView.dequeueReusableCell(withIdentifier: "Cell", for: indexPath )

cell.textLabel?.text = names[indexPath.row]

return cell

}

}

If you’ve ever worked with UITableView, this code should look very familiar. First you return the number of rows in the table as the number of items in your names array.

Next, tableView(\_:cellForRowAt:) dequeues table view cells and populates them with the corresponding string from the names array.

Next, you need a way to add new names so the table view can display them. Implement the addName IBAction method you Ctrl-dragged into your code earlier:

// Implement the addName IBAction

@IBAction func addName(\_ sender: UIBarButtonItem) {

let alert = UIAlertController(title: "New Name", message: "Add a new name", preferredStyle: .alert)

let saveAction = UIAlertAction(title: "Save", style: .default) {

[unowned self] action in

guard let textField = alert.textFields?.first,

let nameToSave = textField.text else {

return

}

self.names.append(nameToSave)

self.tableView.reloadData()

}

let cancelAction = UIAlertAction(title: "Cancel", style: .cancel)

alert.addTextField()

alert.addAction(saveAction)

alert.addAction(cancelAction)

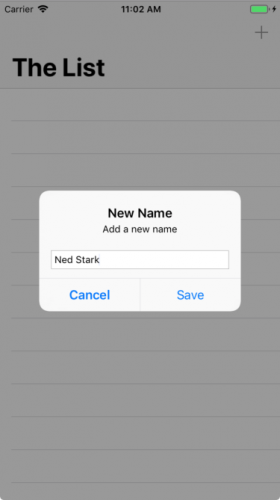
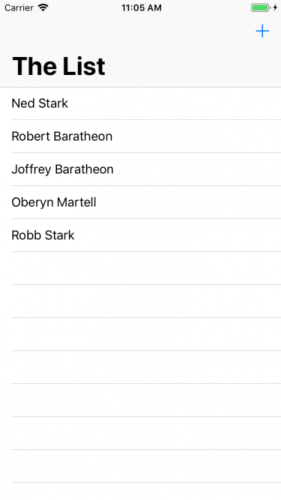
present(alert, animated: true)

}

Every time you tap the **Add** button, this method will present a UIAlertController with a text field and two buttons: **Save** and **Cancel**.

**Save** inserts the text fields current text into the names array then reloads the table view. Since the names array is the model backing the table view, whatever you type into the text field will appear in the table view.

Finally, build and run your app for the first time. Next, tap the **Add** button. The alert controller will look like this:

Add four or five names to the list. You should see something similar to below:

Your table view will display the data and your array will store the names, but the big thing missing here is **persistence**. The array is in memory but if you force quit the app or reboot your device, your hit list will be wiped out.

Core Data provides persistence, meaning it can store data in a more durable state so it can outlive an app re-launch or a device reboot.