iOS Table Views Controllers / Navigation Controllers

Lecture 05

Jonathan R. Engelsma, Ph.D.

TOPICS

- More on View Controllers
- Navigation Controllers
- Table View Overview
- Static Table Views
- Dynamic Table Views

VIEW CONTROLLERS

- We've seen previously that view controllers are the "glue" between models and views. (e.g. looking *inward*)
- However, in addition to managing views & models, they also communicate and coordinate with other view controllers when UI transitions occur within the app. (e.g. looking *outward*)
- Storyboards make it much easier for us to handle these outward looking concerns of view controllers.

VC RESPONSIBILITIES

- Making sure the view gets into the interface! Usually not the managing view controller, but some other view controller…
- Provides animations as views appear/disappear.
- View Controllers work together to save / restore state, which allow the app to restart if terminated or placed in the background.

VC DATA MGMT GUIDELINES

- Unless self-contained, a destination view controller's references to app data should come from the source view controller.
- Use interface builder as much as possible!
- Always use a delegate to communicate back to other view controllers. A view controller should not need to know the type of its source view controller.
- Avoid unnecessary connections to objects external to your view controller.

https://developer.apple.com/library/ios/featuredarticles/ViewControllerPGforiPhoneOS/ManagingDataFlowBetweenViewControllers/ManagingDataFlowBetweenViewControllers.html#//apple_ref/doc/uid/TP40007457-CH8-SW1

ROOT VIEW CONTROLLER

- The view controller that manages the view at the top of the view hierarchy is know as the *root view controller*.
- Responsible for:
  - handling rotation of the user interface
  - manipulation of the status bar, present or not? light or dark styling?
- There is only one root view controller in an app.

SUBORDINATE RELATIONSHIPS

- Apps may consist of additional view controllers that are *subordinate* to the root view controller.
- Subordinate relationships:
  - parentage (containment)
  - presentation (formally known as modal views)

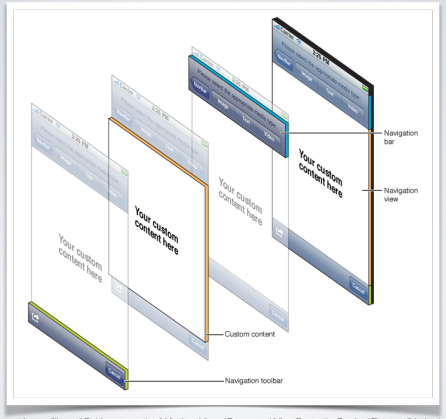
PARENTAGE

- A view controller (parent) may contain a subordinate view controller (child).
- A child view controller's view, if visible is a *subview* of the parent's view.
- The parent view controller makes the views of its children visible, or replaces with other views.

NAVIGATION CONTROLLERS

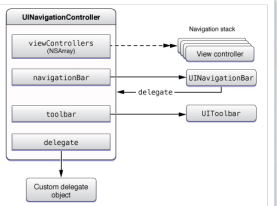
- Navigation controllers manage a stack of view controllers.
- Provide a drill-down interface for hierarchical content.
- Each view controller in the stack manages a distinct view.
- The navigation controller navigates within the stack of view controllers.

UINavigationController



https://developerapple.com/library/iOs/documentation/WindowsViews/ConceptualViewControllerCatalog/Chapters/NavigationControllers.html

NAV OBJECTS



- Stack is a LIFO collection of view controllers.
- First item added to the stack is known as the **root view controller**.

HOW TO USE

- Using UINavigation Controller with Storyboards
  1. Drag a UINavigationController from the object library to the storyboard.
  2. Cntrl-Click drag from nav controller to the "root" view.
  3. Release click and select "root view"
  4. In subsequent segues, make sure the type of the segue is set to push.

PRESENTATION RELATIONSHIP

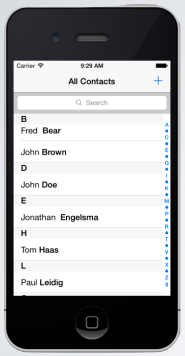
- A view controller (*presenting VC*) can present another view controller (*presented VC*)
- The presented VC is NOT a child.
- The presented VC's view covers part or all of the presenting VC's view.
- Formally known (iOS 4 and earlier) as a *modal view controller*.

TABLE VIEWS IN IOS

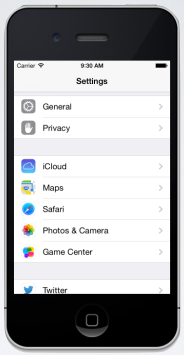
- Table views have many purposes:
  - To let users navigate through hierarchically structured data
  - To present an indexed list of items
  - To display detail information and controls in visually distinct groupings
  - To present a selectable list of options

UITABLEVIEW

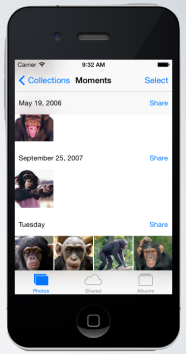
- Multiple vertical rows, but only one column.
- Static or dynamic.
- Scrollable (inherits UIScrollView)
- Customized via a datasource and delegate protocol.
- Lots of different prefabricated layouts for the individual cells in our UITableViews.



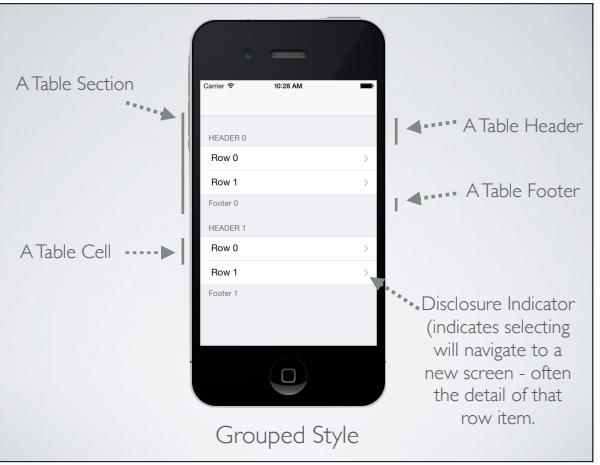
Dynamic / Plain Indexed List



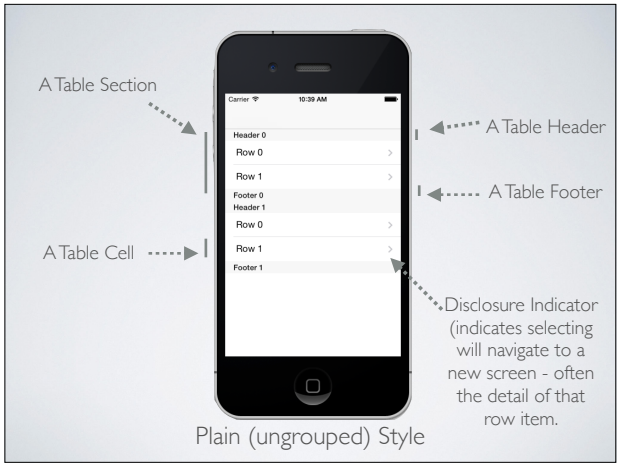
Static / Default Layout / Grouped



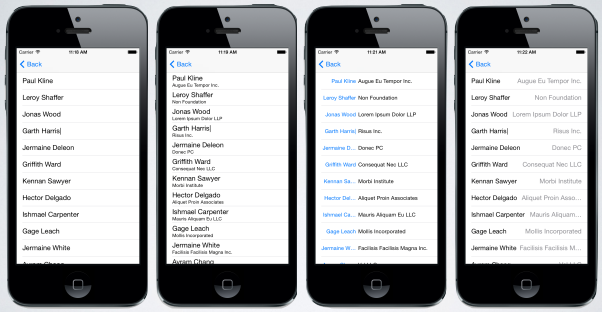
Dynamic / Custom Layout



Grouped Style



CELL STYES



CONTROLLERS

- Most of the time we will manage UITableViews with UITableViewControllers.
- Normally, the UITableView fills the entire view.
- Since the UITableView paradigm involves selecting a row to "zoom" in on the details of that row, we need a way to navigate back and forth between the table view and the detail view.
- This is normally done with a Navigation Controller.

STATIC TABLE DEMO

DEMO!!

DYNAMIC TABLE VIEWS

- Static table views are great for use cases where the data is not dynamic, such as setting screens, etc.
- If we are displaying data dynamically (e.g. displaying data resulting from a DB query, or fetch to a web API.) we need to use a different approach.
- We need to programmatically provide the UITableView with its data!

UITABLEVIEW PROTOCOLS

- UITableView's delegate and dataSource properties:
  - dataSource (implements UITableViewDataSource): Mediates the apps model data and the table view hierarchy (e.g. specifies cells, headers, rows, etc.)
  - delegate (implements UITableViewDelegate): manages how the tableview will be displayed, row selection, etc.

UITableView Protocols

- When we create a UITableViewController in Interface Builder:
  - it has a property tableView that points to its UITableView object.
  - it automatically wires the controller object up to be the delegate and dataSource.

UITableViewDataSource

- Three important methods on UITableViewDataSource:
  - numberOfSectionsInTableView: how many sections are in the table?
  - tableView:numberOfRowsInSection: how many rows are in the given section?
  - tableView:cellForRowAtIndexPath: provide a IUTableViewCell for a given position within the table.

Dynamic Table Demo

DEMO!!

UITableViewDelegate

- The UITableView delegate handles a number of details concerning how the table view will look.
- Handles what happens when a row is selected.
- We can also use it to edit the rows in a table view.

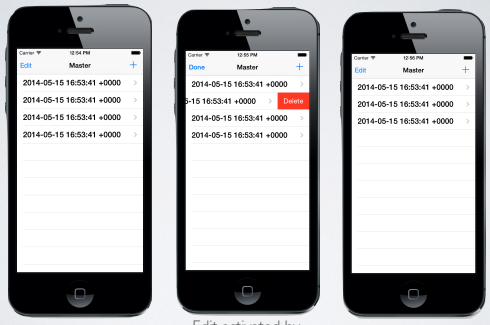


Table with 4 rows

Edit activated by swiping from right to left on the second row.

After delete is pressed.

READING ASSIGNMENT

- Chapter 8: Programming iOS 8 (by Neuburg)

ROW EDIT DEMO

DEMO!!