iPad

iOS App Development
Fall 2010 — Lecture 25

Questions?

Announcements

- Nothing newsworthy — should be working on final assignments

Today's Topics

- iPad overview
- iPad-specific templates
- Modal view presentation styles
- Popovers
- Split views
- Universal apps

Notes

- I’m showing the relevant portions of the view controller interfaces and implementations in these notes
- Remember to release relevant memory in the -dealloc methods — they are not shown here
- You will also need to wire up outlets and actions in IB
- Where delegates or data sources are used, they too require wiring in IB

iPad Overview

Physical Specs

- 9.7 inch (diagonal) touch screen
  - Supports 11 simultaneous touch points
- Accelerometer
- Assisted GPS (on 3G versions)
- WiFi
- Bluetooth
- No cameras
- No gyroscope

iPad Screen

	Portrait	Landscape	Pixel Density
iPhone 4	640 x 960 px	960 x 640 px	326 PPI
iPad	768 x 1024 px	1024 x 768 px	132 PPI
Other iOS devices	320 x 480 px	480 x 320 px	163 PPI

iPad Human Interface Guidelines

- Consider using popovers for some modal tasks
- Migrate toolbar content to the top
- Reduce full-screen transitions
- Flatten hierarchies

iPhone & iPod touch Apps

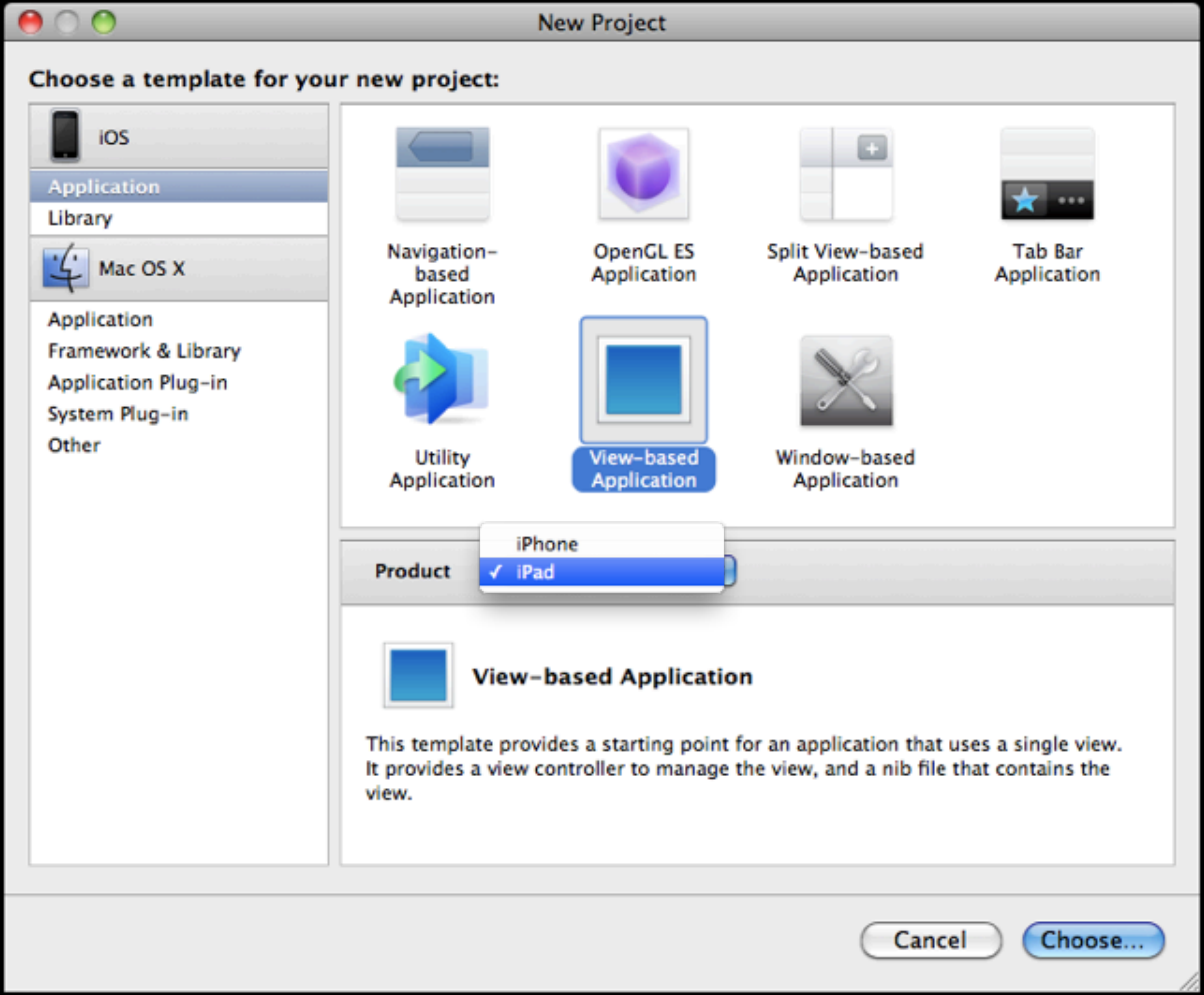
- By default, the iPad will run apps designed for iPhone or iPod touch at either 1x or 2x size

iPad Related Templates

iPad Related Project Templates

- You may have noticed that there are several built-in iOS templates which allow you to choose between iPhone and iPad...
  - OpenGL ES
  - Tab Bar
  - View Based
  - Window Based

iPad Related Project Templates



Product: iPhone [✓ iPad]

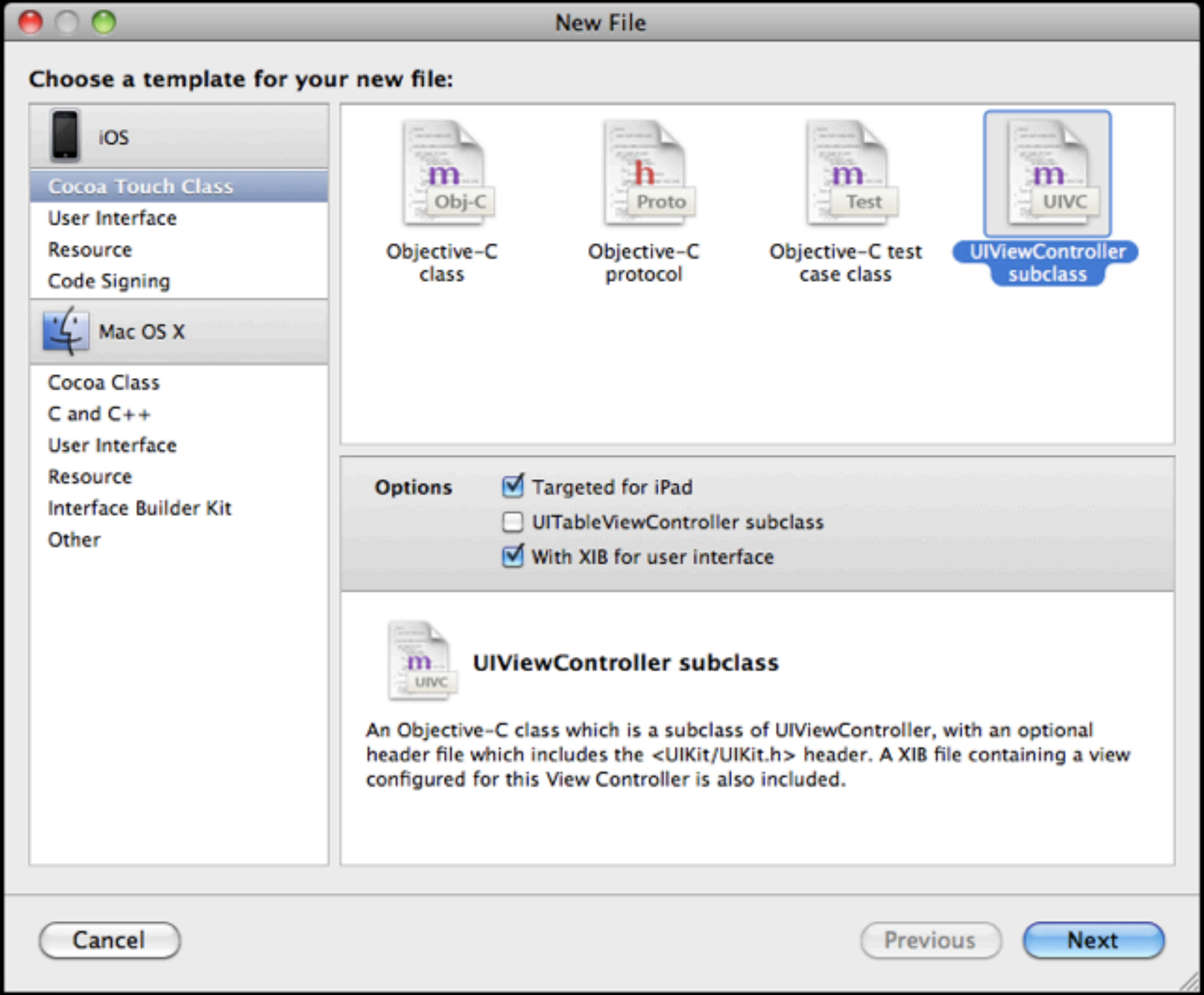
View-based Application

This template provides a starting point for an application that uses a single view. It provides a view controller to manage the view, and a nib file that contains the view.

iPad Related File Templates

- There are also several places where you can create a new file that’s specifically geared toward iPad...
  - Cocoa Touch Class → UIViewController subclass
  - User Interface → various NIB templates

iPad Specific App Templates



Choose a template for your new file:

Options
- [✓] Targeted for iPad
- [ ] UITableViewController subclass
- [✓] With XIB for user interface

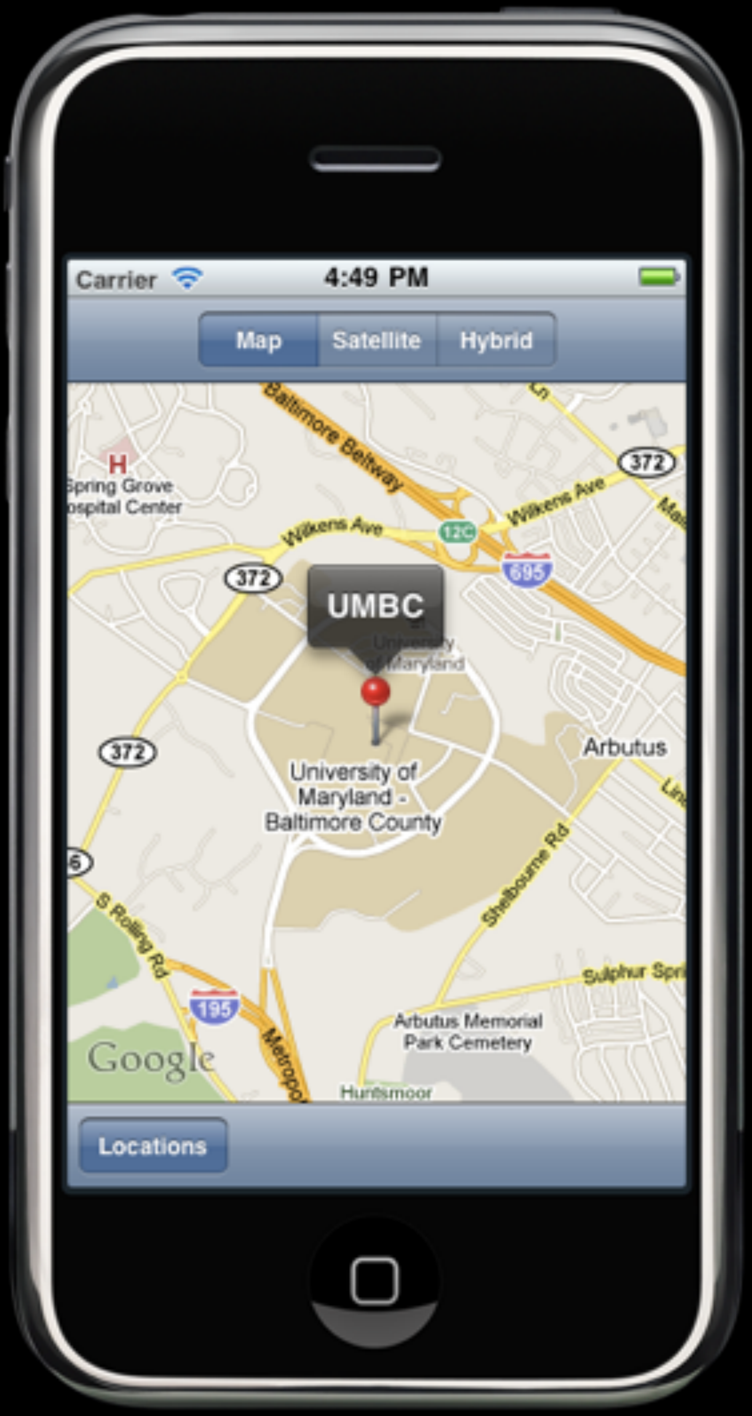
UIViewController subclass

An Objective-C class which is a subclass of UIViewController, with an optional header file which includes the <UIKit/UIKit.h> header. A XIB file containing a view configured for this View Controller is also included.

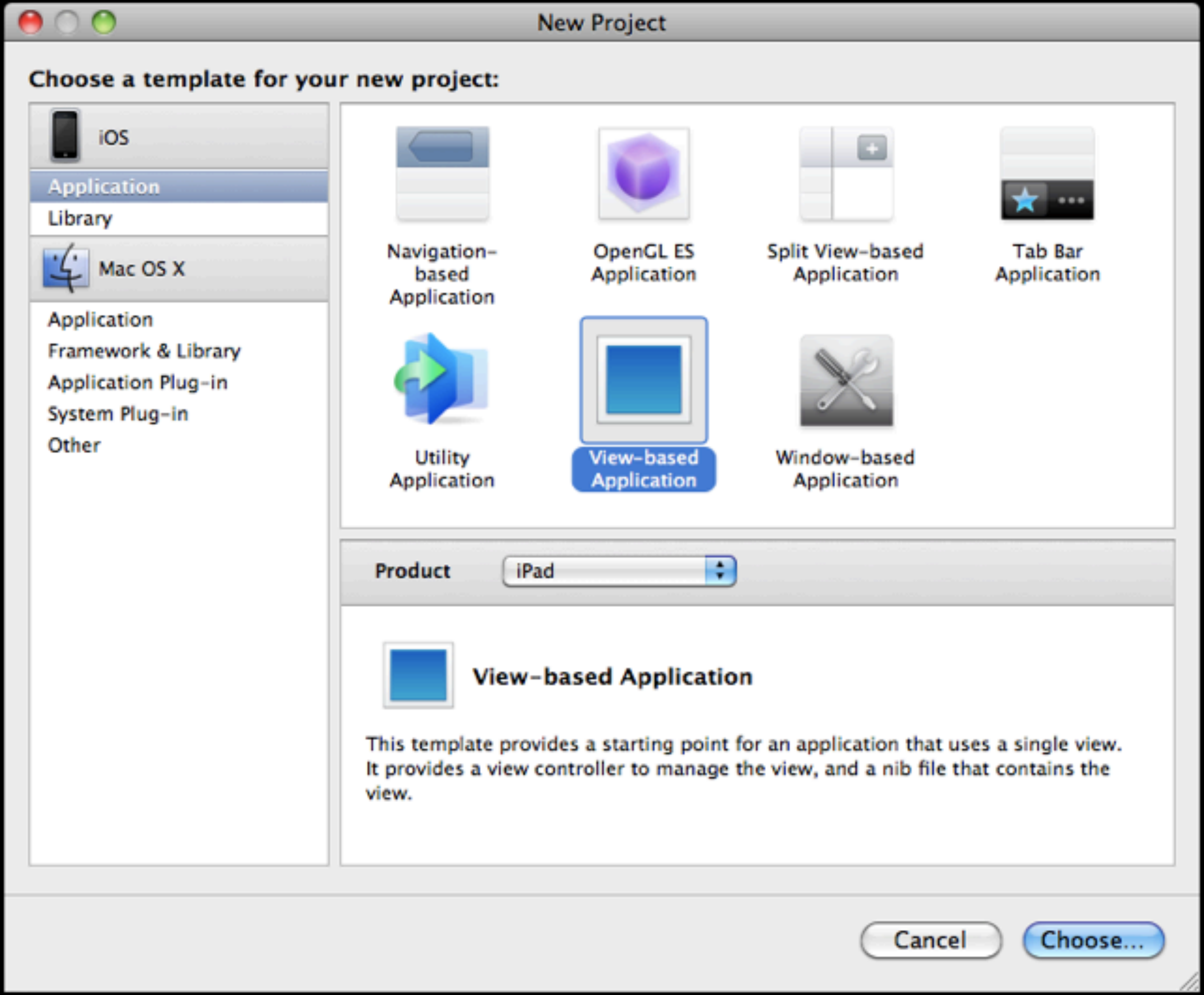
View Based Example

University Map Example

- Remember the University Map example from the Core Location & MapKit lecture?
- Let's recreate this app for iPad
- To do so, let's start by utilizing the View-based Application Template



New View-based App for iPad



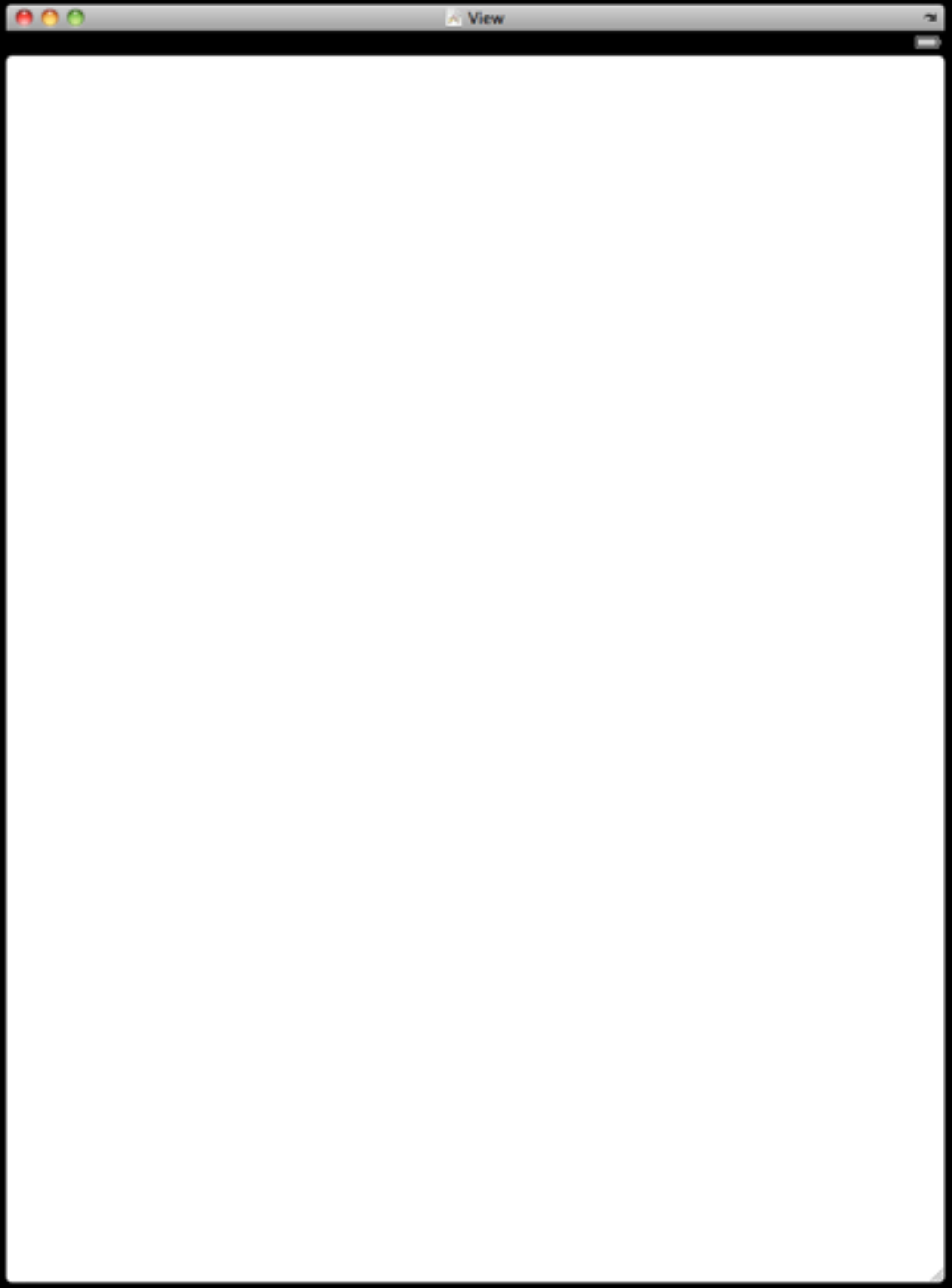
Product: iPad

View-based Application

This template provides a starting point for an application that uses a single view. It provides a view controller to manage the view, and a nib file that contains the view.

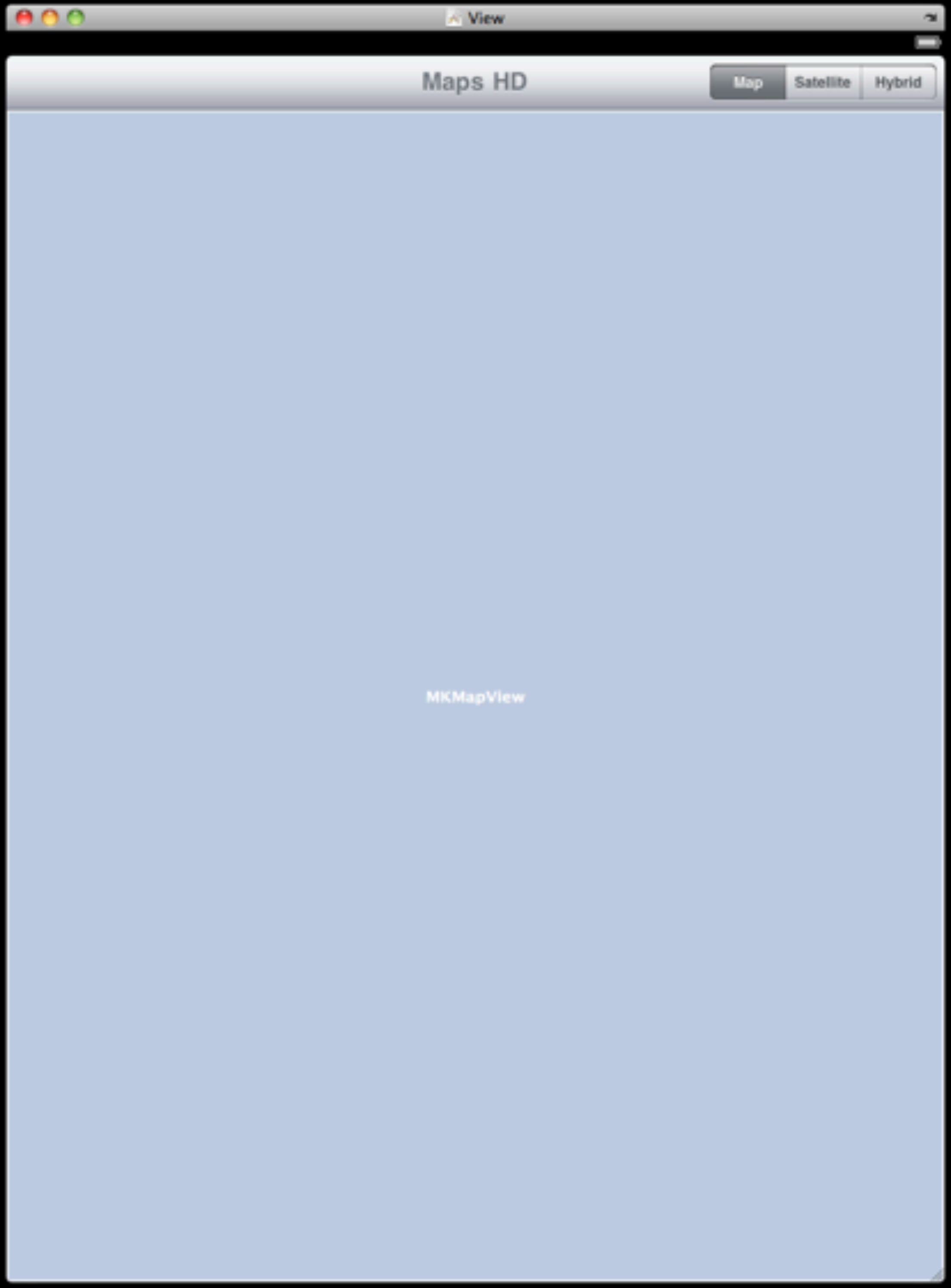
View-based App Template

- If you look around the *.[mh] classes that were stubbed out you'll notice that everything is fairly similar to a normal iPhone app template
- However, once you open the *.xib files, you'll notice an immediate difference...
  - It's sized for the iPad's screen resolution



MapsHDViewController.xib

- Let's start with just...
  - A toolbar with some segmented controls
  - A large UIMapView



MapsHDViewController.h

```
#import <UIKit/UIKit.h>
#import <MapKit/MapKit.h>

@interface MapsHDViewController : UIViewController {

}

@property (nonatomic, retain) IBOutlet MKMapView *map;

- (IBAction)updateMapType:(id)sender;

@end
```

MapsHDViewController.m

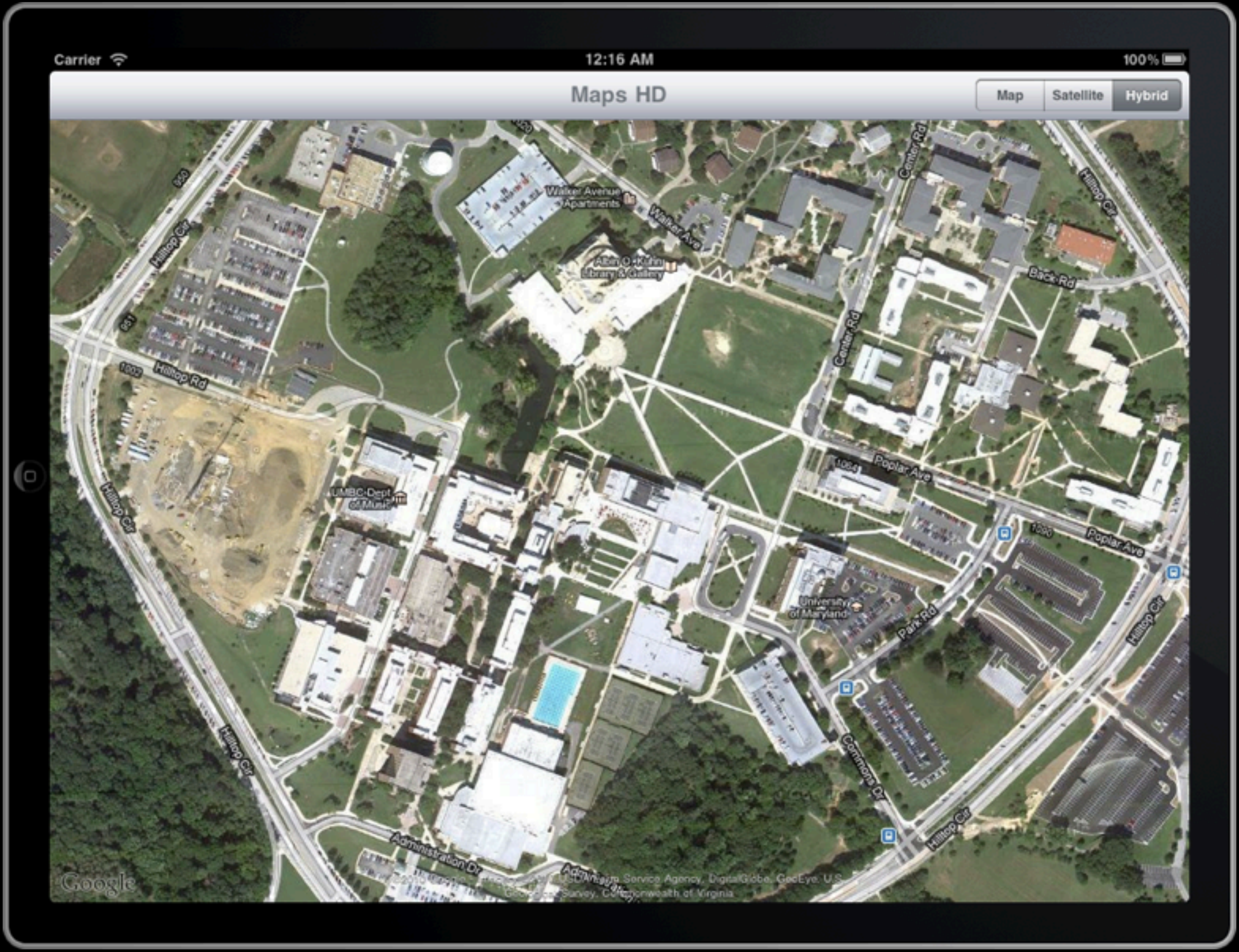
```objc
#import "MapsHDViewController.h"

@implementation MapsHDViewController

@synthesize map;

- (IBAction)updateMapType:(id)sender {
    switch ([sender selectedSegmentIndex]) {
       case 0:
          self.map.mapType = MKMapTypeStandard;
          break;
       case 1:
          self.map.mapType = MKMapTypeSatellite;
          break;
       default:
          self.map.mapType = MKMapTypeHybrid;
          break;
    }
}

/* ... */
```

MapsHDViewController.m

```
/* ... */

- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)
                                                                 interfaceOrientation {
    return YES;
}

@end
```

This method is commented out on iPhone based templates, for iPad it defaults to all orientations

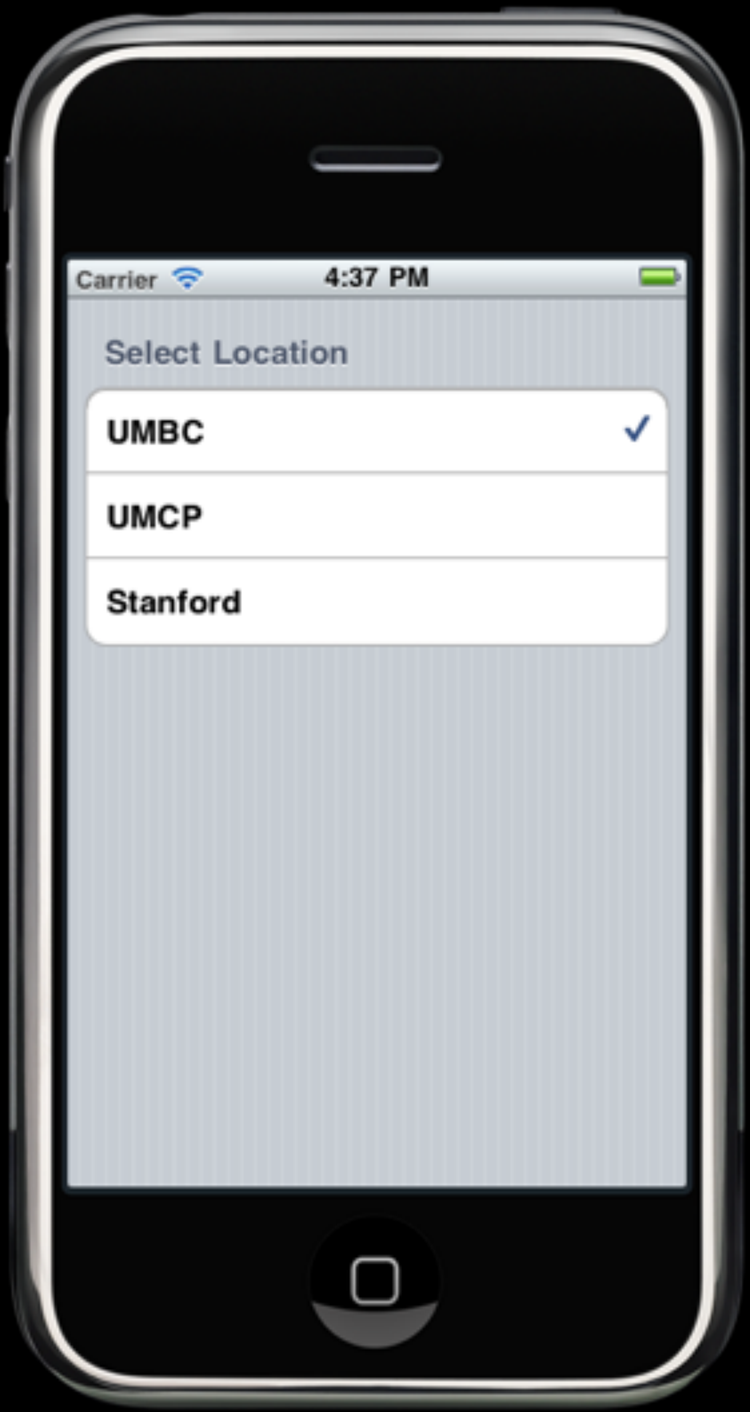
The Resulting App



iPad Modal View Presentation Styles

Modal Dialogs

- Remember in our iPhone version of the University Map app, we displayed a modal dialog
- Once they made a choice, the map was updated to drop a pin at the new location



Select Location

UMBC ✓

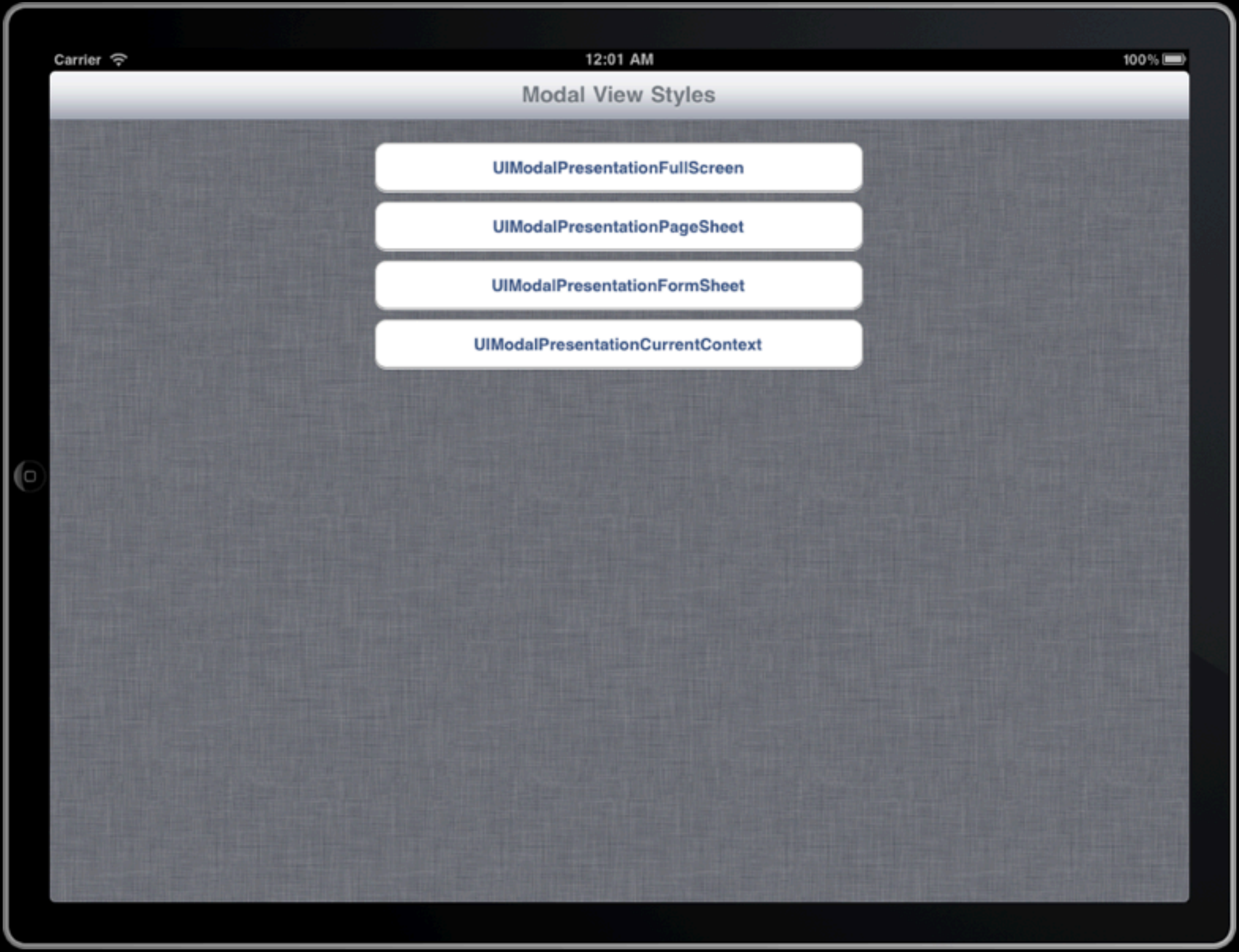
UMCP

Stanford

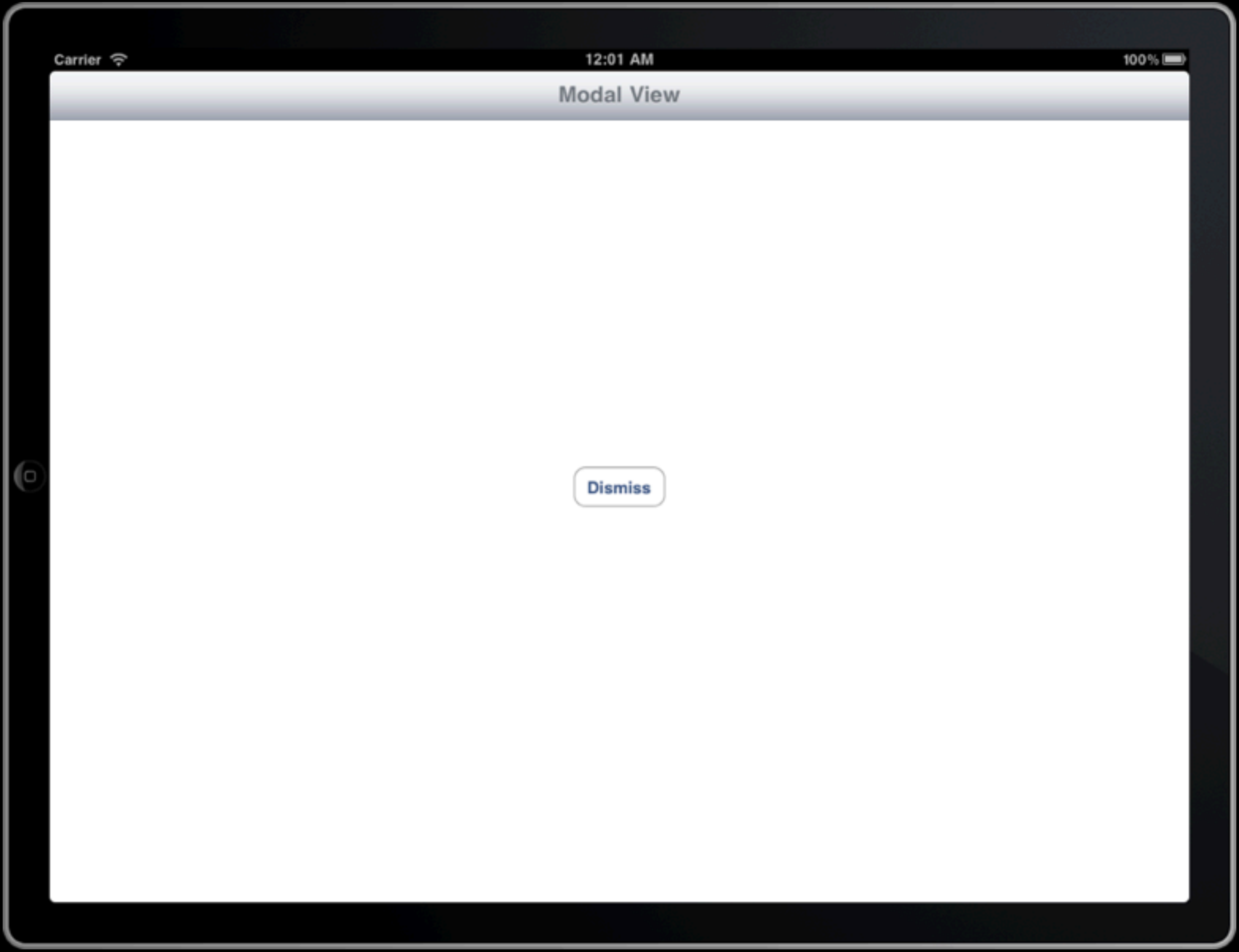
Modal Presentation

- In the modal dialog examples we previously examined the modal view consumed the entirety of the screen
  - This usually makes sense on the iPhone & iPod touch as screen real estate is at a premium
- However, given the larger display on the iPad there are several modal presentation styles

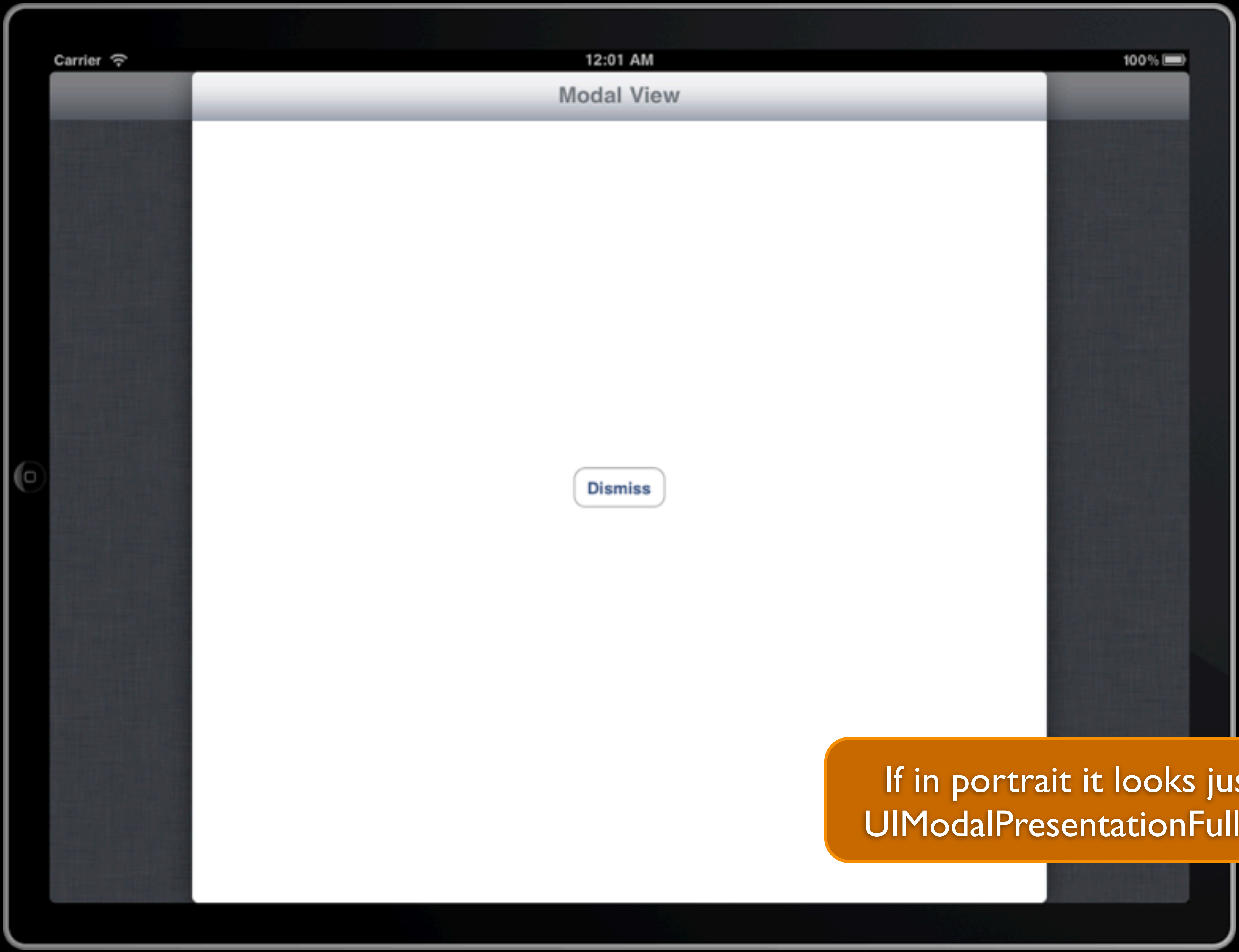
Our Normal View


Carrier
12:01 AM
100%
Modal View Styles
UIModalPresentationFullScreen
UIModalPresentationPageSheet
UIModalPresentationFormSheet
UIModalPresentationCurrentContext

UIModalPresentationFullScreen

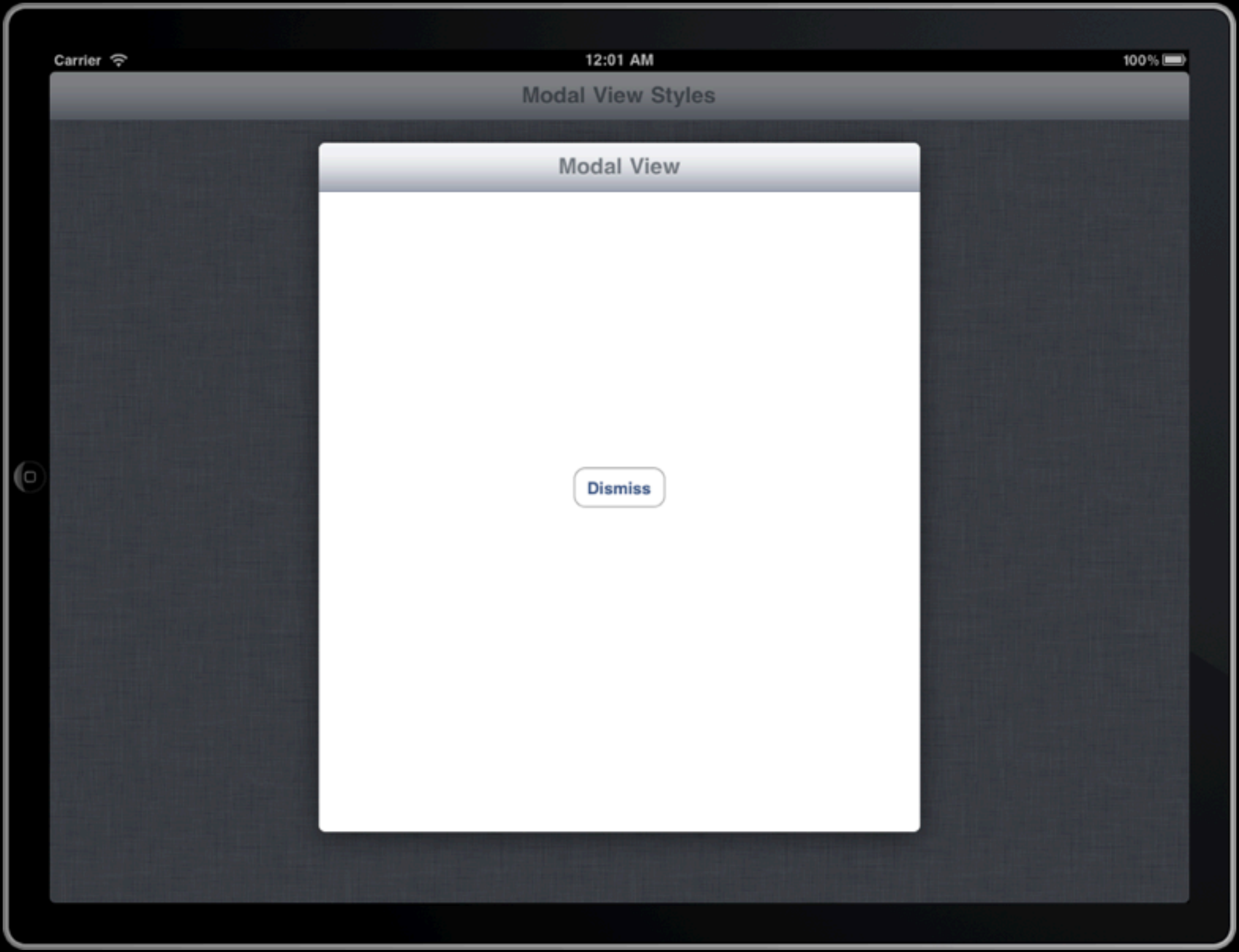


UIModalPresentationPageSheet

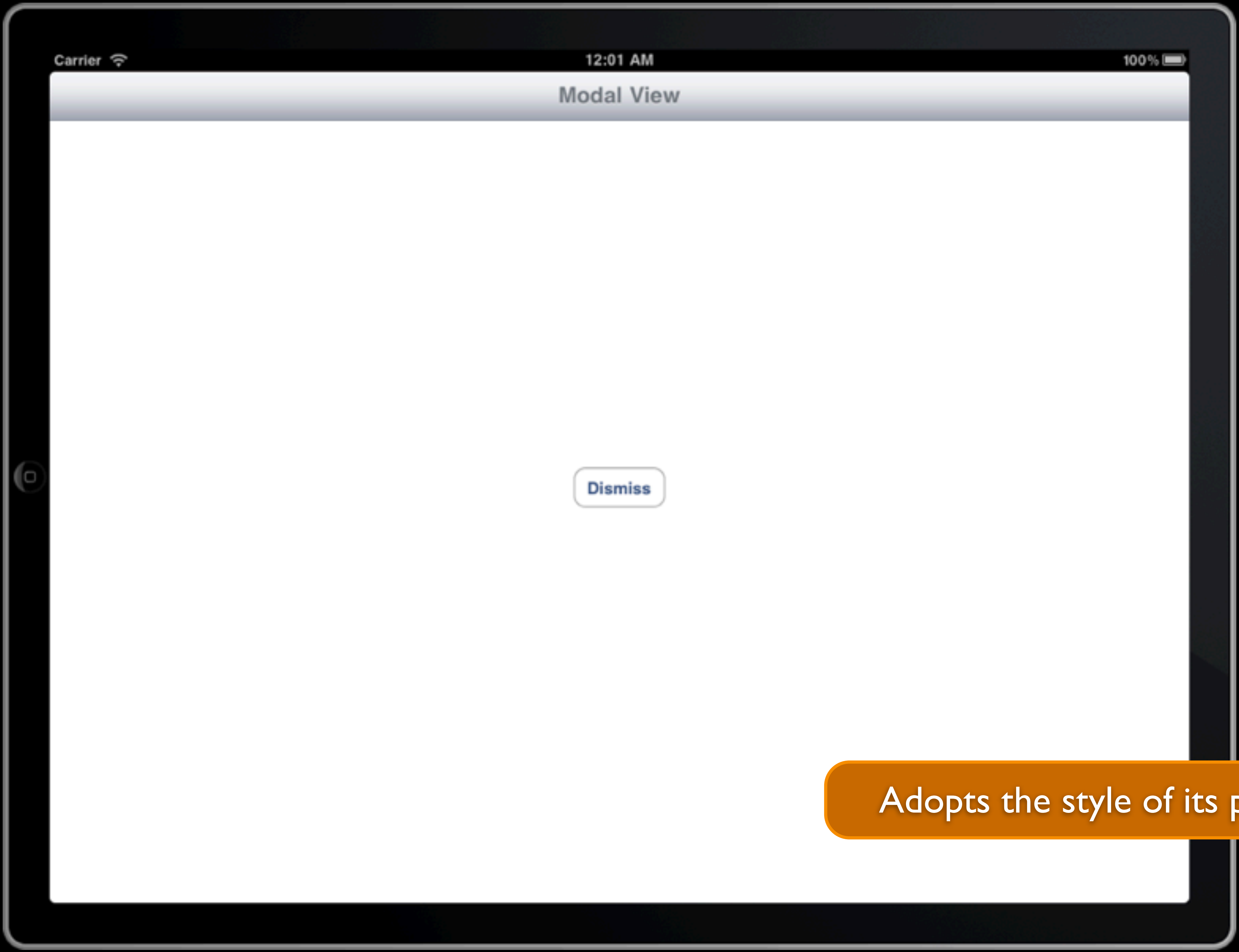


If in portrait it looks just like UIModalPresentationFullScreen

UIModalPresentationFormSheet



UIModalPresentationCurrentContext



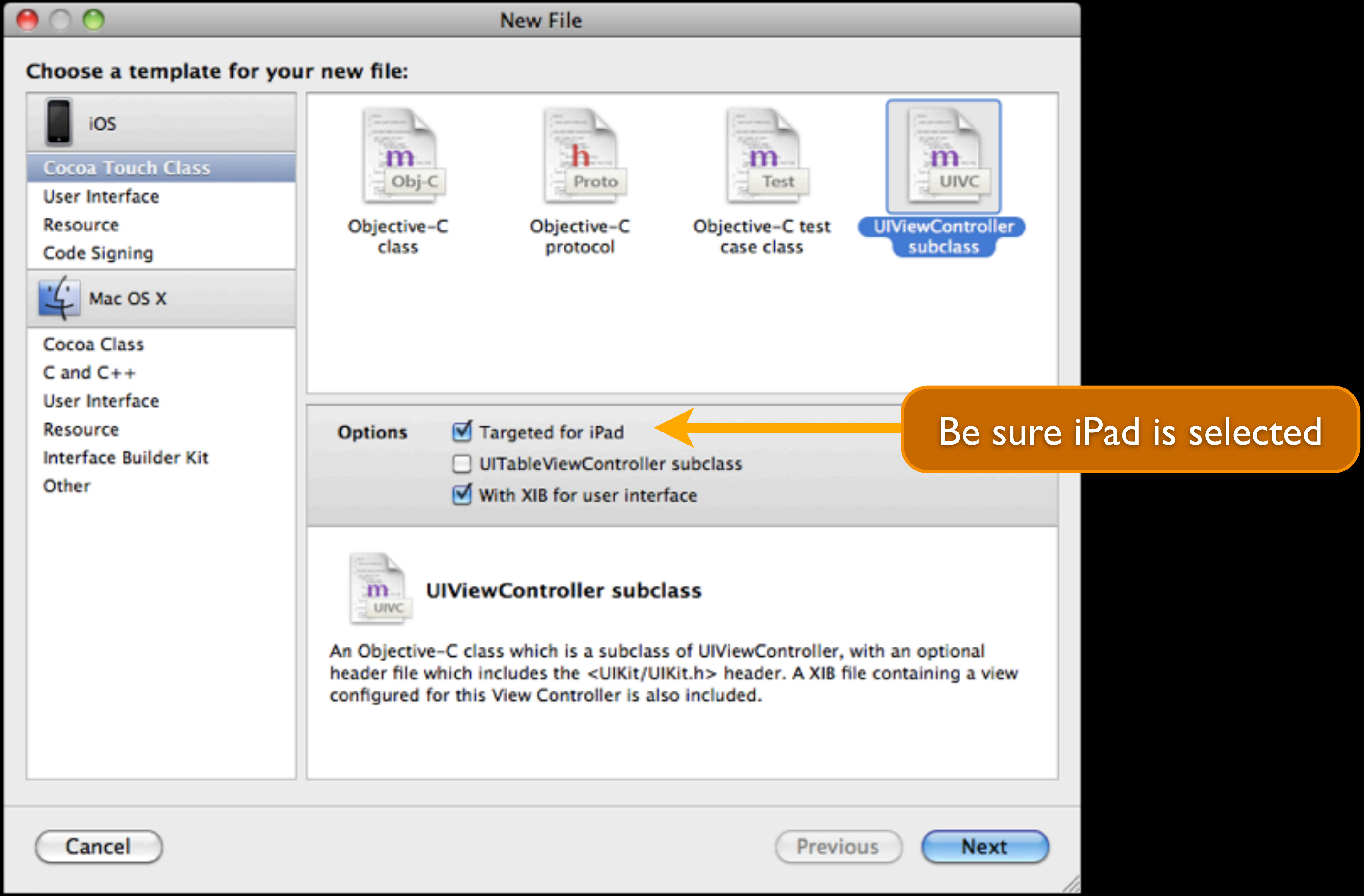
Adopts the style of its parent

Modal View Example

Displaying the Modal Dialog

- Let’s add the university selection component back into our app as a modal form sheet
  - Create a universities view controller to display modally
    - Need to create *.{m,h,xib}
    - Wire up table data source & delegate methods
    - Need way to signal selection back to the caller
  - Add button to open modal dialog into main view
    - Create action method
    - Receive callback once selection is made
    - Dismiss modal dialog

Creating the Universities View Controller



New File

Choose a template for your new file:

Options
- [✓] Targeted for iPad
- [ ] UITableViewController subclass
- [✓] With XIB for user interface

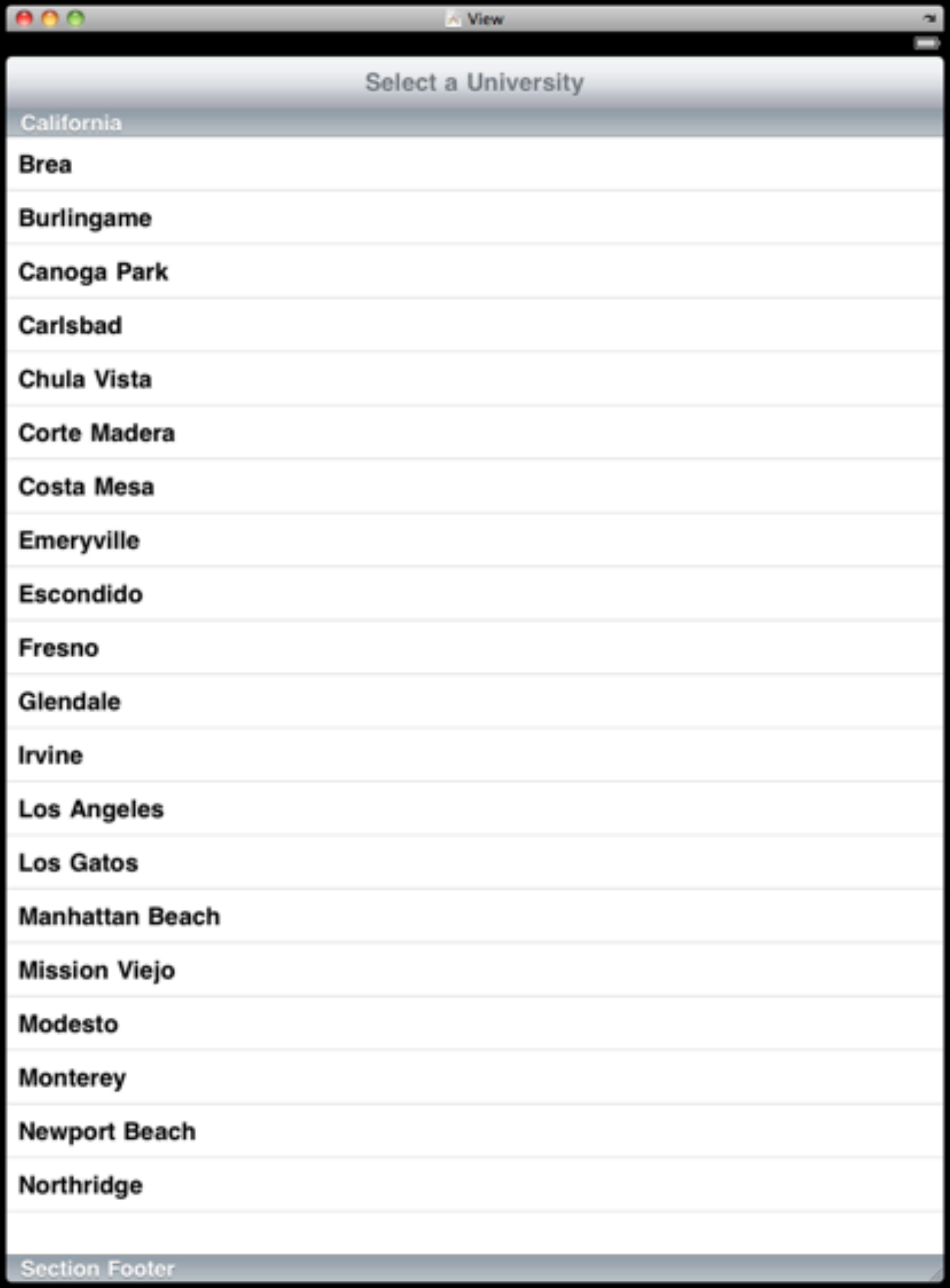
Be sure iPad is selected

UIViewController subclass

An Objective-C class which is a subclass of UIViewController, with an optional header file which includes the <UIKit/UIKit.h> header. A XIB file containing a view configured for this View Controller is also included.

UniversitiesViewController.xib

- Added a nav bar at the top
- Added a table view as the main body
- Wired up table delegate and data sources back to the view controller


View
Select a University
California
Brea
Burlingame
Canoga Park
Carlsbad
Chula Vista
Corte Madera
Costa Mesa
Emeryville
Escondido
Fresno
Glendale
Irvine
Los Angeles
Los Gatos
Manhattan Beach
Mission Viejo
Modesto
Monterey
Newport Beach
Northridge
Section Footer

UniversitiesViewController.h

```
#import <UIKit/UIKit.h>

// Forward declaration of corresponding delegate
@protocol UniversitiesViewControllerDelegate;

// The view controller class
@interface UniversitiesViewController : UIViewController <UITableViewDelegate,
                                                          UITableViewDataSource> {
    id<UniversitiesViewControllerDelegate> delegate;
}

@property(nonatomic, retain) NSArray *universities;
@property(nonatomic, assign) int selectedIndex;
@property(nonatomic, assign) id<UniversitiesViewControllerDelegate> delegate;

@end

// The delegate for the view controller
@protocol UniversitiesViewControllerDelegate
@required
- (void)selectedUniversityFromController:(UniversitiesViewController *)controller;
@end
```

We're going to use a delegate to inform the caller when a university has been selected

Delegates are usually stored as "weak links"

UniversitiesViewController.m

```
#import "UniversitiesViewController.h"
#import "University.h"

@implementation UniversitiesViewController

@synthesize delegate, universities, selectedIndex;

#pragma mark -
#pragma mark View lifecycle

- (void)viewDidLoad {
    [super viewDidLoad];
    self.modalPresentationStyle = UIModalPresentationFormSheet;
    self.universities = [NSArray arrayWithObjects:
                    [University universityWithTitle:@"UMBC"
                                  latitude:39.2551 longitude:-76.7110],
                    [University universityWithTitle:@"UMCP"
                                  latitude:38.9916 longitude:-76.9431],
                    [University universityWithTitle:@"Stanford"
                                  latitude:37.427297 longitude:-122.170372],
                    nil];
}

/* ... */
```

UniversitiesViewController.m

```
/* ... */

#pragma mark -
#pragma mark Table view data source

- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
    return 1;
}


- (NSInteger)tableView:(UITableView *)tableView
 numberOfRowsInSection:(NSInteger)section {
    return [self.universities count];
}

/* ... */
```

UniversitiesViewController.m

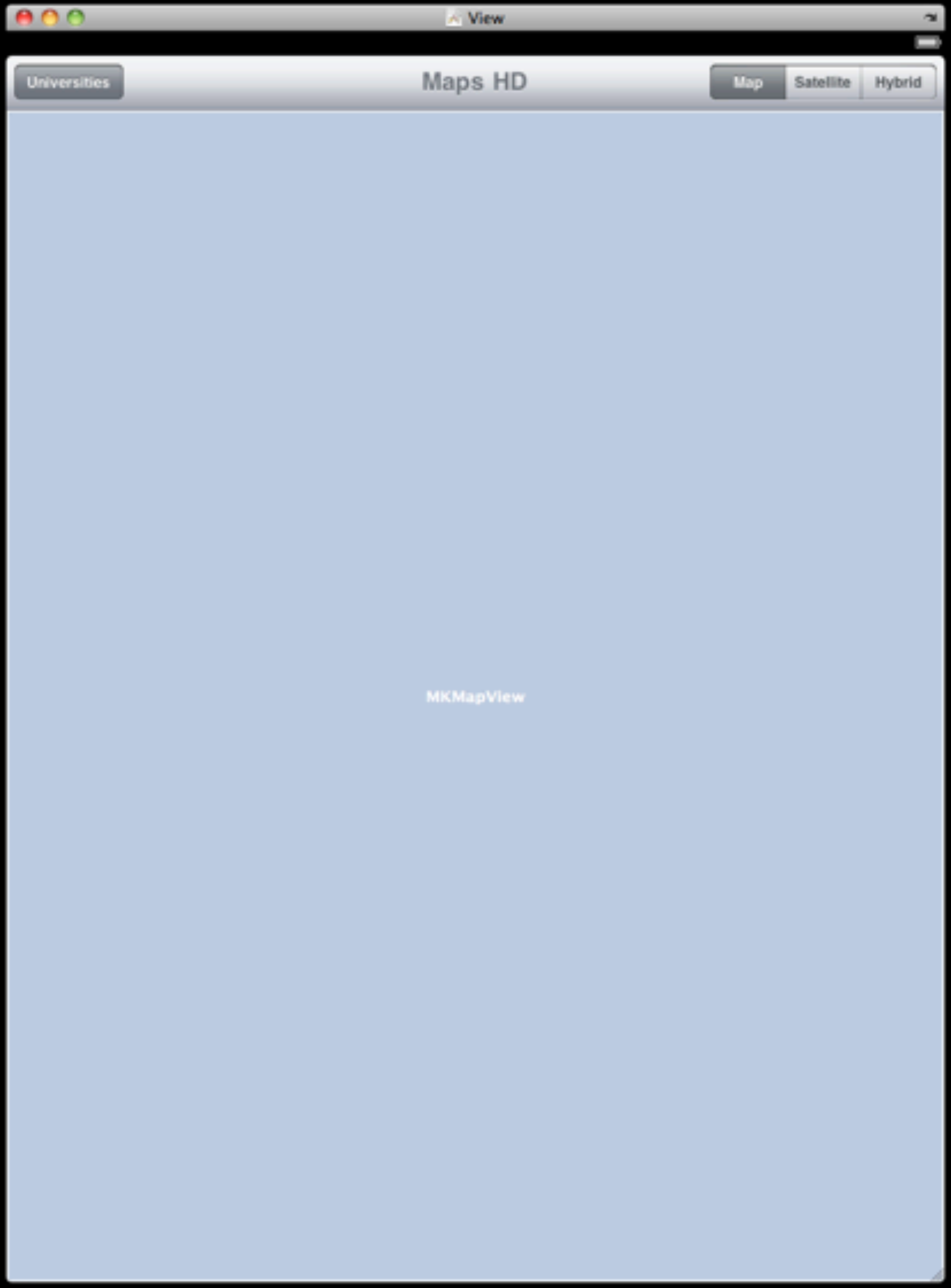
```
/* ... */

- (UITableViewCell *)tableView:(UITableView *)tableView
         cellForRowAtIndexPath:(NSIndexPath *)indexPath {

  static NSString *CellIdentifier = @"Cell";

  UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier];
  if (cell == nil) {
    cell = [[[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault
                                   reuseIdentifier:CellIdentifier] autorelease];
  }

  // Configure the cell...
  cell.textLabel.text = [[self.universities objectAtIndex:indexPath.row] title];
  if (self.selectedIndex == indexPath.row) {
    cell.accessoryType = UITableViewCellAccessoryCheckmark;
  } else {
    cell.accessoryType = UITableViewCellAccessoryNone;
  }

  return cell;
}

/* ... */
```

UniversitiesViewController.m

```
/* ... */

#pragma mark -
#pragma mark Table view delegate

- (void)tableView:(UITableView *)tableView
        didSelectRowAtIndexPath:(NSIndexPath *)indexPath {
    self.selectedIndex = indexPath.row;
    [tableView reloadData];
    [self.delegate selectedUniversityFromController:self];
}

/* ... */

@end
```

MapsHDViewController.xib

- The added button wired up to call the appropriate action method



MapsHDViewController.h

```
#import <UIKit/UIKit.h>
#import <MapKit/MapKit.h>
#import "UniversitiesViewController.h"

@interface MapsHDViewController : UIViewController
                                  <UniversitiesViewControllerDelegate> {

}

@property (nonatomic, retain) IBOutlet MKMapView *map;
@property (nonatomic, retain) UniversitiesViewController *universitiesView;

- (IBAction)updateMapType:(id)sender;
- (IBAction)selectLocation;

@end
```

To store the university controller

Action method to bring up modal dialog

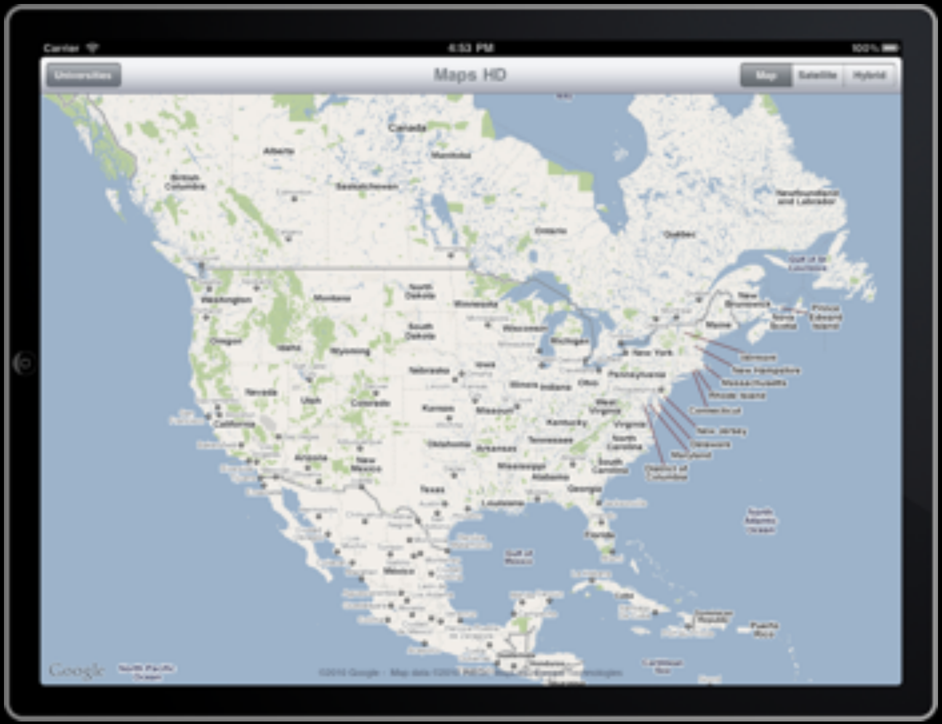
MapsHDViewController.m

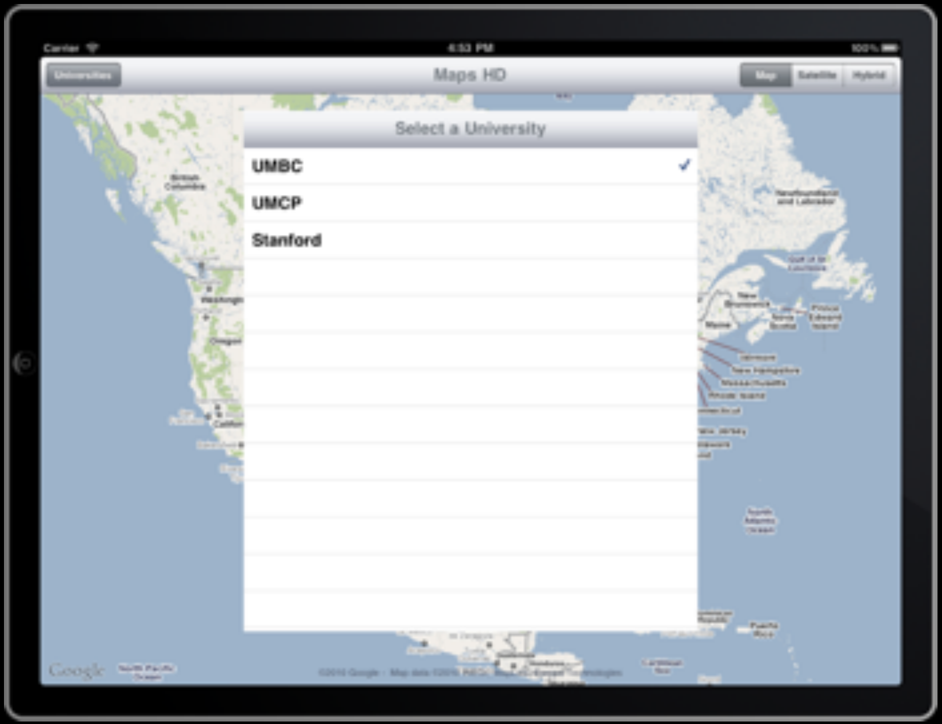
```
#import "MapsHDViewController.h"
#import "UniversitiesViewController.h"
#import "University.h"

@implementation MapsHDViewController

@synthesize map, universitiesView;

#pragma mark -
#pragma mark UniversitiesViewControllerDelegate method

- (void)selectedUniversityFromController:(UniversitiesViewController *)controller {

  [controller dismissModalViewControllerAnimated:YES];

  University *university = [controller.universities
                            objectAtIndex:controller.selectedIndex];
  [self.map setRegion:MKCoordinateRegionMake(university.coordinate,
                                    MKCoordinateSpanMake(.015, .015)) animated:YES];

  // empty all annotations, add the current university
  [self.map removeAnnotations: [self.map annotations]];
  [self.map addAnnotation:university];
  [self.map selectAnnotation:university animated:YES];
}
/* ... */
```

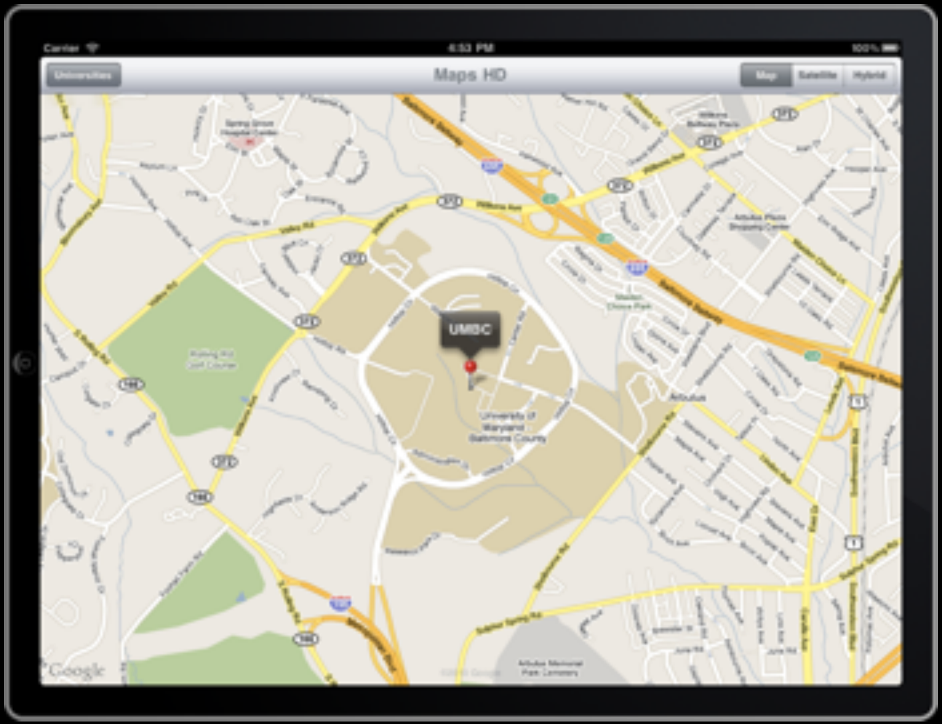
MapsHDViewController.m

```
/* ... */

- (IBAction)selectLocation:(id)sender {
    [self presentModalViewController:self.universitiesView animated:YES];
}

/* ...other methods remain the same as before... */

@end
```

The Resulting App







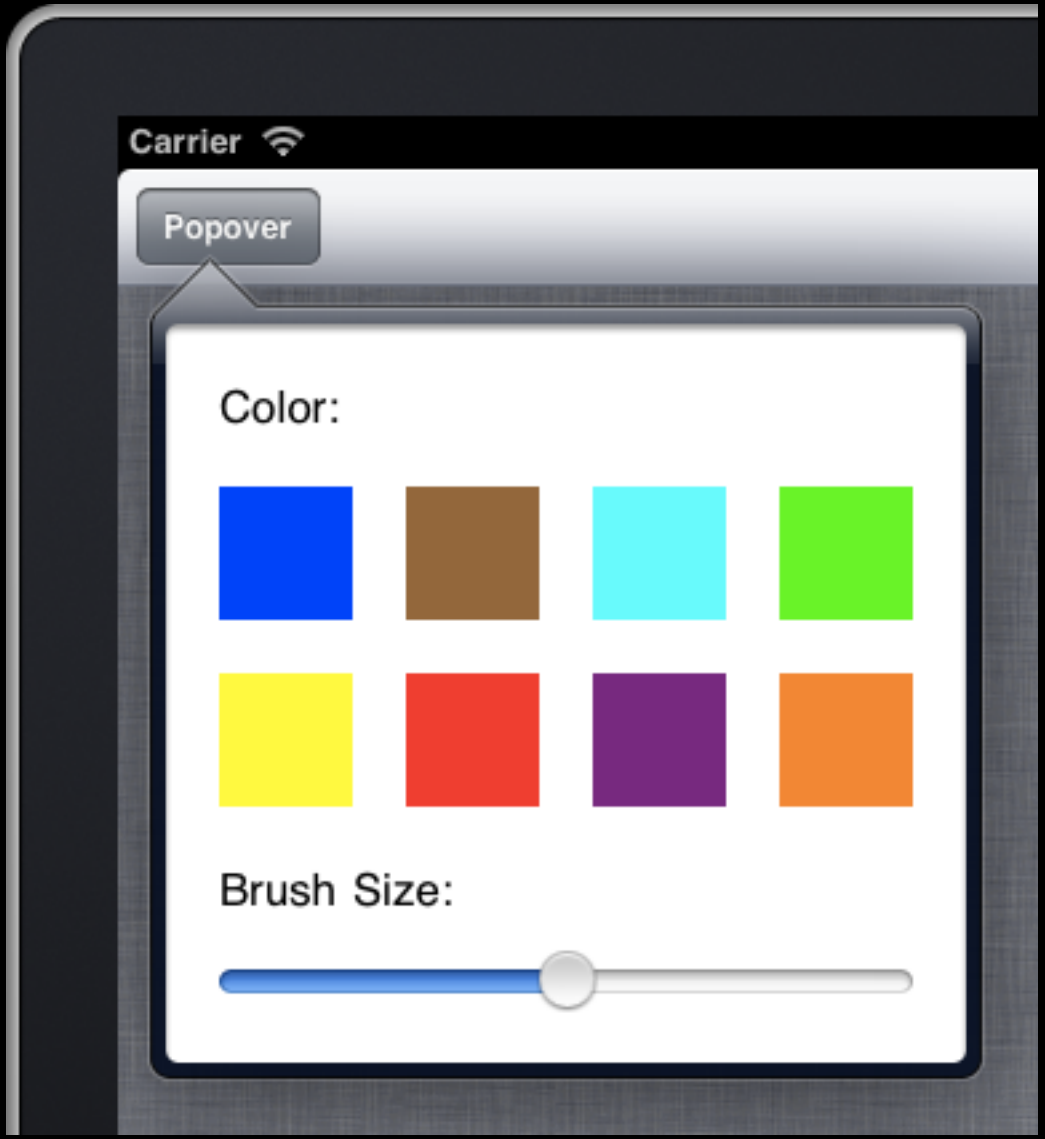
Observations

- In this case much of the space occupied by the modal view is wasted...
  - Just a few rows with actual data
  - Text takes up just a small amount of the horizontal space
  - Totally takes the user away from the map to ask a question
- Let’s look at a different widget that addresses these issues

Popovers

Popovers

- Added in iOS 3.2 exclusively for the use by iPad
- Presents a view temporarily in a less obtrusive way that doesn't take over the screen like a modal view
- Popovers automatically manage dismissing the popover when tapping off of the widget



UIPopoverController

- The UIPopoverController class is used to create and display popovers
- It isn’t a view controller in the traditional sense that you’d subclass it and add content
- Instead you’ll provide it a view controller to display...

```
- (id)initWithContentViewController:(UIViewController *)viewController;
```

- Frequently we’ll also set the size of the popover...

```
@property (nonatomic) CGSize popoverContentSize;
```

UIPopoverController

- There are 2 different presentation methods to trigger the opening of a popover
- One allows you to open the popover over a given rectangle on the screen...

```
- (void)presentPopoverFromRect:(CGRect)rect
                        inView:(UIView *)view
      permittedArrowDirections:(UIPopoverArrowDirection)arrowDirections
                      animated:(BOOL)animated;
```

- The other is a convenience method for the common use case of opening a popover button from a toolbar

```
- (void)presentPopoverFromBarButtonItem:(UIBarButtonItem *)item
               permittedArrowDirections:(UIPopoverArrowDirection)arrowDirections
                               animated:(BOOL)animated;
```

UIPopoverArrowDirection

- UIPopoverArrowDirection is an enum of acceptable directions for the popover arrow to be pointing...

```
enum {
    UIPopoverArrowDirectionUp = 1UL << 0,
    UIPopoverArrowDirectionDown = 1UL << 1,
    UIPopoverArrowDirectionLeft = 1UL << 2,
    UIPopoverArrowDirectionRight = 1UL << 3,
    UIPopoverArrowDirectionAny = UIPopoverArrowDirectionUp |
                                 UIPopoverArrowDirectionDown |
                                 UIPopoverArrowDirectionLeft |
                                 UIPopoverArrowDirectionRight,
    UIPopoverArrowDirectionUnknown = NSIntegerMax
};
typedef NSUInteger UIPopoverArrowDirection;
```

- Can be combined together to support multiple directions by using the bitwise OR operator

UIPopoverControllerDelegate

- Popovers can also have a delegate which can be used to get callbacks for the following 2 events...

```
- (BOOL)popoverControllerShouldDismissPopover:(UIPopoverController *)popoverController;
- (void)popoverControllerDidDismissPopover:(UIPopoverController *)popoverController;
```

Popover Example

Displaying the Popover

- Let’s change our modal dialog to instead be a popover...
  - Should remove the title bar from the university view
  - Need to instantiate the popover and wrap our university view inside of it
  - Should probably set the size of the popover view
  - Remove unnecessary display modal display/dismiss code
  - Open popover on button click

MapsHDViewController.h

```
#import <UIKit/UIKit.h>
#import <MapKit/MapKit.h>
#import "UniversitiesViewController.h"

@interface MapsHDViewController : UIViewController
                                  <UniversitiesViewControllerDelegate> {

}

@property (nonatomic, retain) IBOutlet MKMapView *map;
@property (nonatomic, retain) UniversitiesViewController *universitiesView;
@property (nonatomic, retain) UIPopoverController *popover;

- (IBAction)updateMapType:(id)sender;
- (IBAction)selectLocation:(id)sender;

@end
```

Added popover view controller

Added sender argument to this action method (remember to re-wire in IB)

MapsHDViewController.m

```
#import "MapsHDViewController.h"
#import "UniversitiesViewController.h"
#import "University.h"

@implementation MapsHDViewController

@synthesize map, universitiesView, popover;

#pragma mark -
#pragma mark UniversitiesViewControllerDelegate method

- (void)selectedUniversityFromController:(UniversitiesViewController *)controller {

  University *university = [controller.universities
                             objectAtIndex:controller.selectedIndex];
  [self.map setRegion:MKCoordinateRegionMake(university.coordinate,
                             MKCoordinateSpanMake(.015, .015)) animated:YES];

  // empty all annotations, add the current university
  [self.map removeAnnotations: [self.map annotations]];
  [self.map addAnnotation:university];
  [self.map selectAnnotation:university animated:YES];
}

/* ... */
```

Removed modal dismiss call

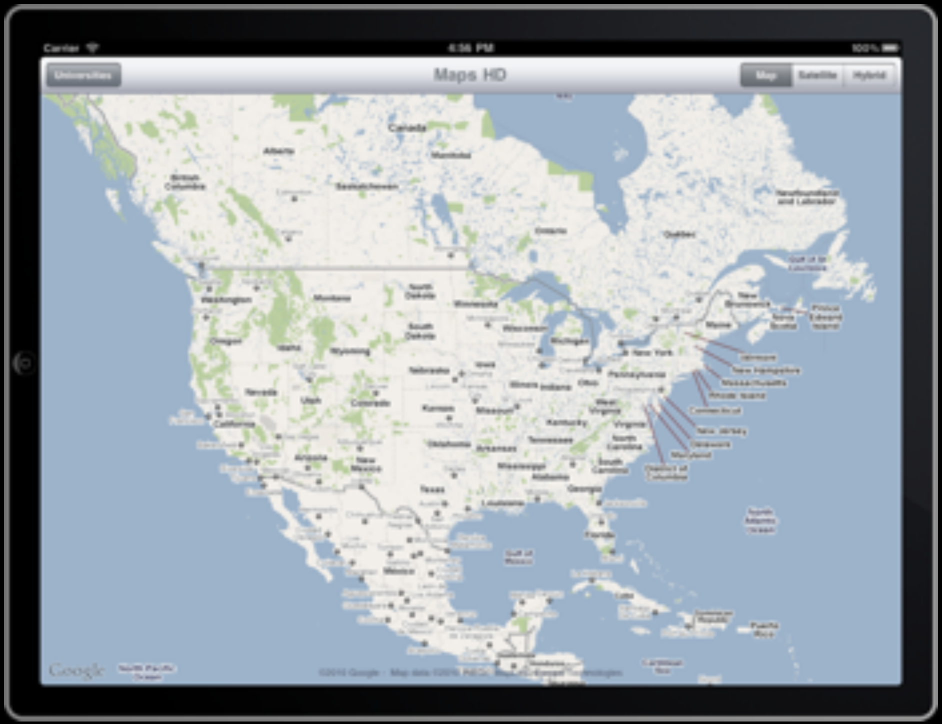
MapsHDViewController.m

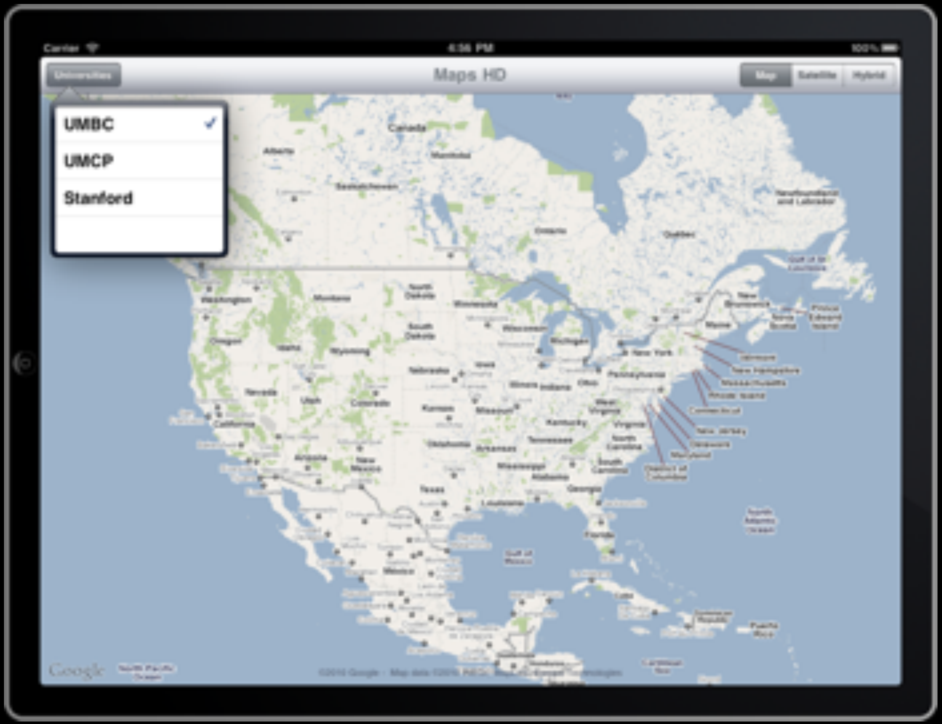
```
/* ... */

- (IBAction)selectLocation:(id)sender {
  [self.popover presentPopoverFromBarButtonItem:sender
                       permittedArrowDirections:UIPopoverArrowDirectionUp
                                       animated:YES];
}

- (void)viewDidLoad {
  [super viewDidLoad];
  self.map.region = MKCoordinateRegionMake(CLLocationCoordinate2DMake(40.0, -95.0),
                                           MKCoordinateSpanMake(20.0, 40.0));
  self.universitiesView = [[[UniversitiesViewController alloc]
                            initWithNibName:@"UniversitiesViewController" bundle:nil]
                            autorelease];
  self.universitiesView.delegate = self;

  self.popover = [[[UIPopoverController alloc]
                   initWithContentViewController:self.universitiesView] autorelease];
  self.popover.popoverContentSize = CGSizeMake(200, 175);
}

/* ...rest of the methods the same... */

@end
```

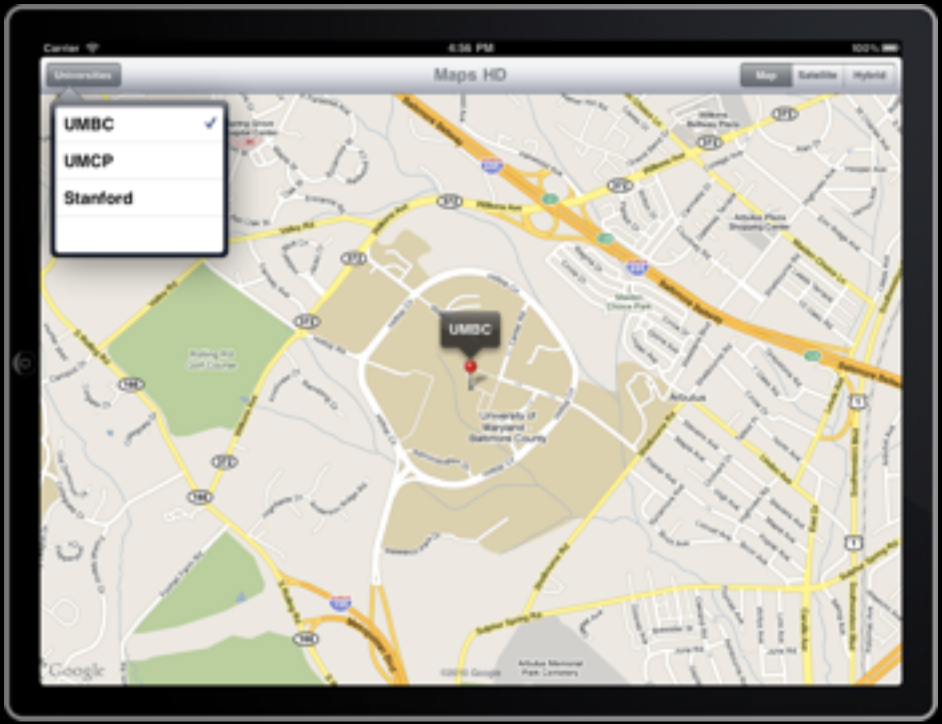
Open popover

Create popover wrapping university controller inside, and set the size

The Resulting App





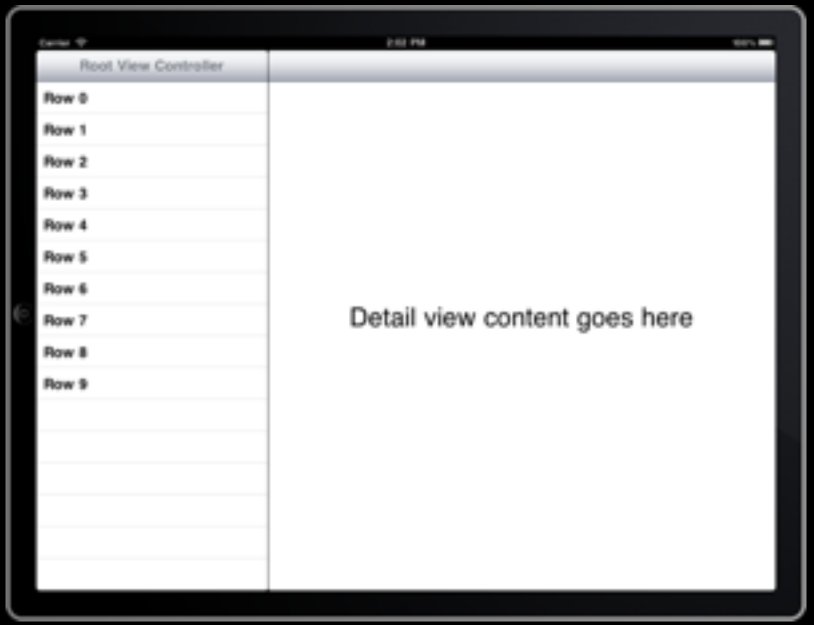


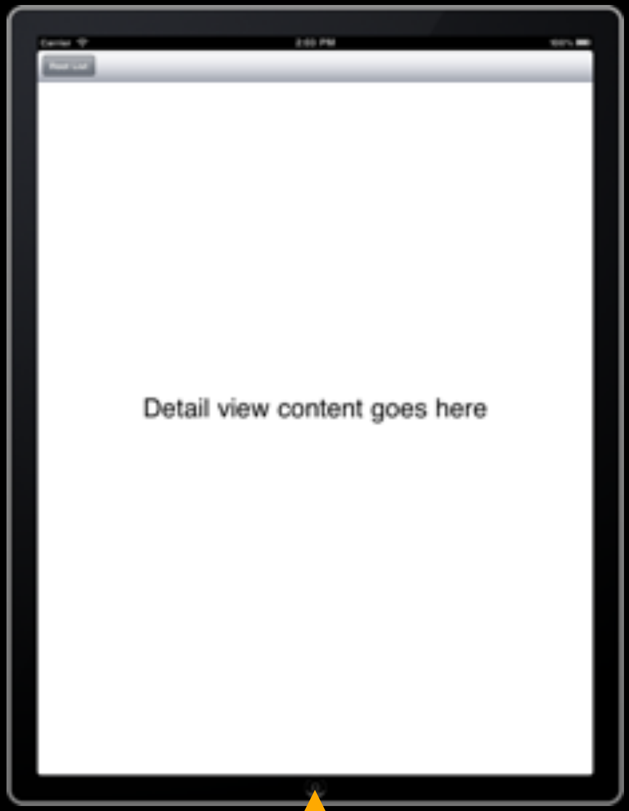
Split Views

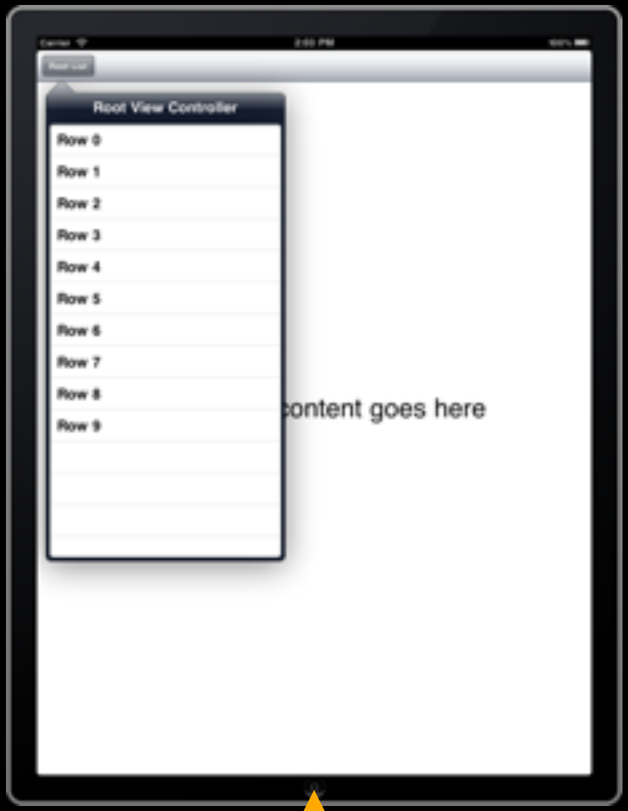
Split Views

- Split view controllers are able of maintaining 2 panes
- In landscape orientation...
  - The view is split with the left pane having a fixed width of 320 px and the right pane occupying the remaining space
  - This is the only configuration, you cannot split top/bottom or adjust the location of the split
- In Portrait orientation..
  - Only the right (detailed) pane is displayed
  - The options that were on the left get placed into a popover that can be activated from the toolbar

Split Views





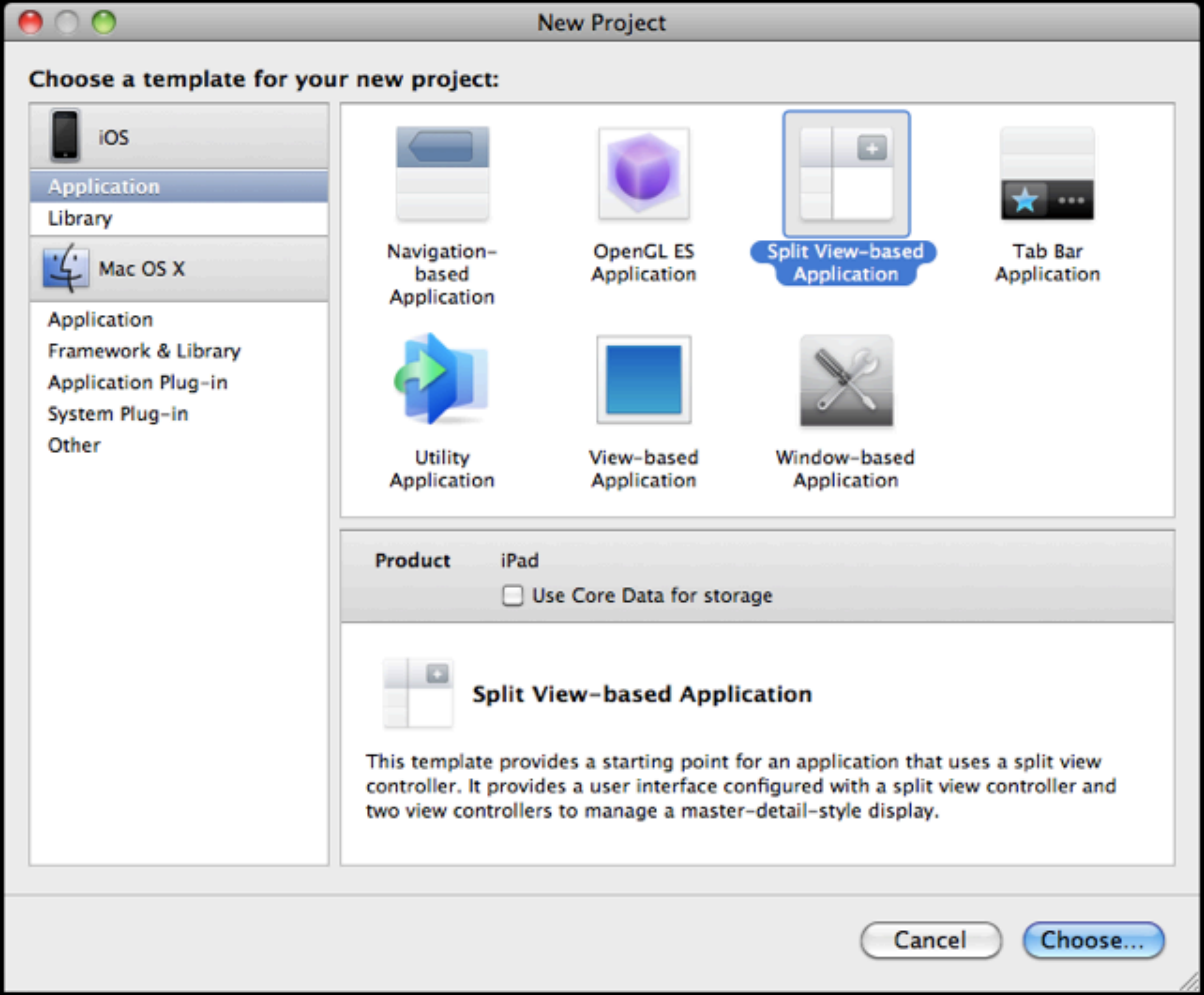


Landscape

Rotated

Tap to access

New Split View-based App for iPad



Choose a template for your new project:

Product iPad

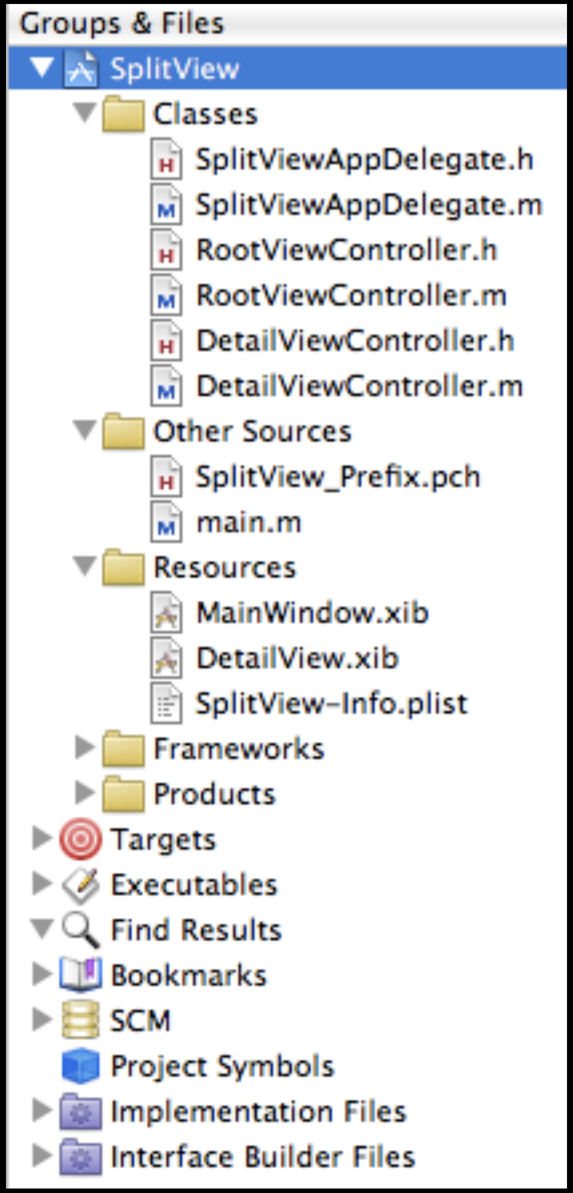
☐ Use Core Data for storage

Split View-based Application

This template provides a starting point for an application that uses a split view controller. It provides a user interface configured with a split view controller and two view controllers to manage a master-detail-style display.

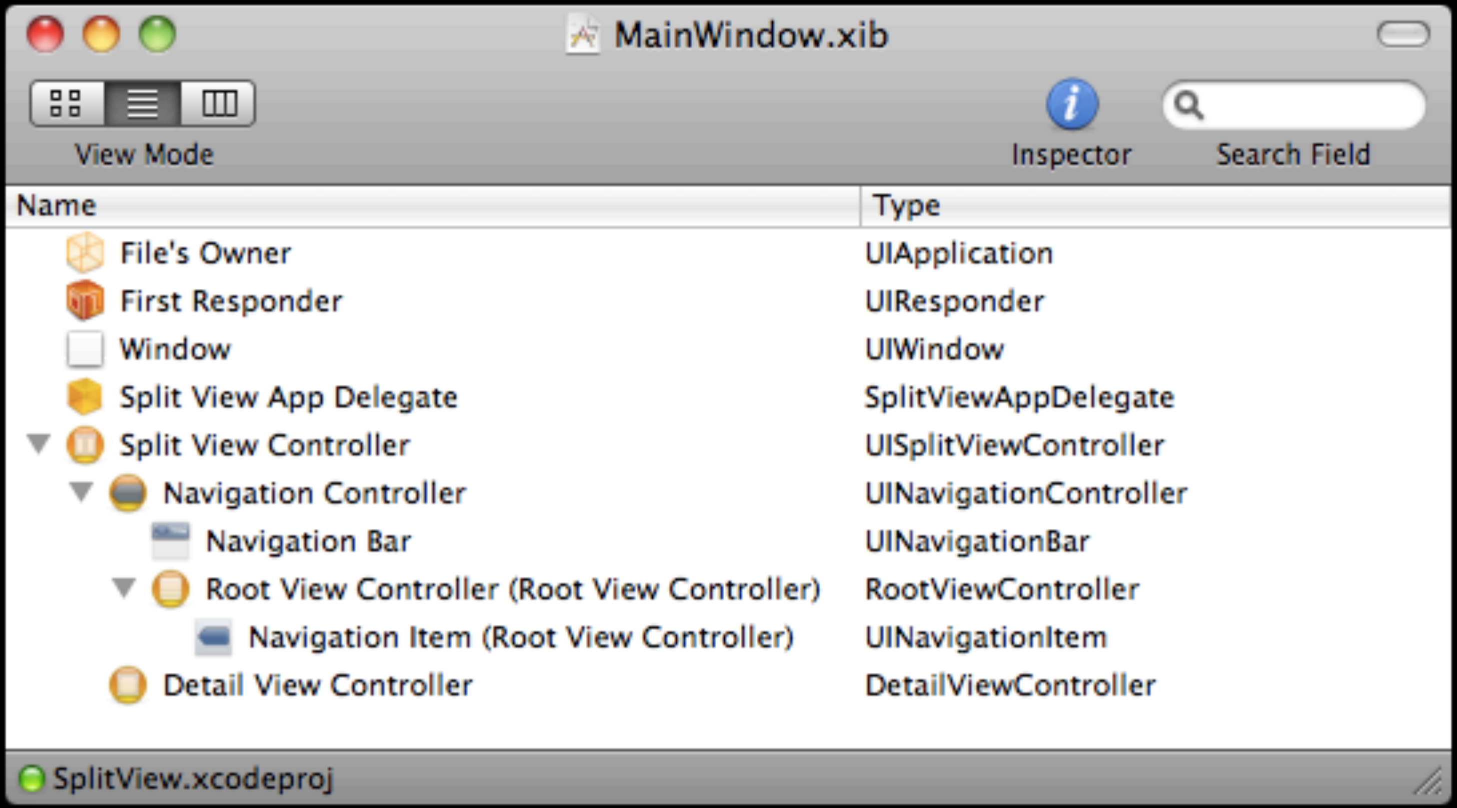
Template Files

- Perhaps the files created by the Split-view based template shouldn't come as a surprise
- For the left pane...
  - RootViewController.{m,h,xib}
- For the right (detailed) pane...
  - DetailedViewController.{m,h,xib}



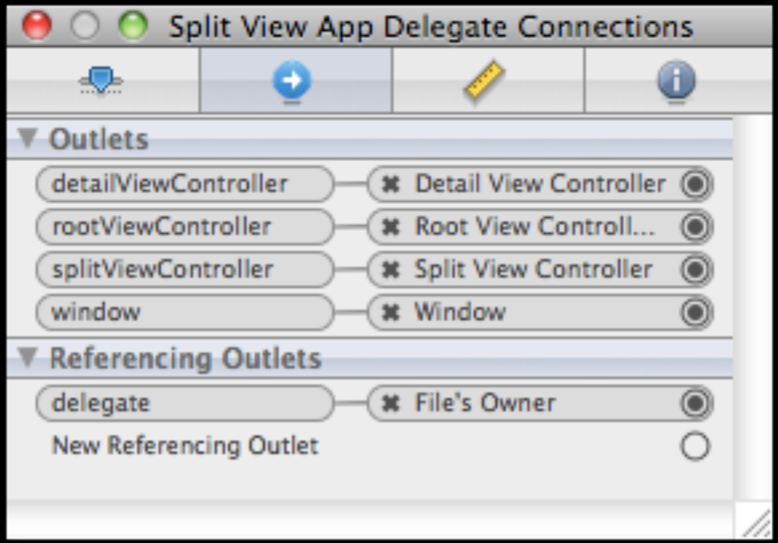
MainWindow.xib

- If you open the main window NIB you'll see how this is configured and wired up...



Name	Type
File's Owner	UIApplication
First Responder	UIResponder
Window	UIWindow
Split View App Delegate	SplitViewAppDelegate
Split View Controller	UISplitViewController
Navigation Controller	UINavigationController
Navigation Bar	UINavigationBar
Root View Controller (Root View Controller)	RootViewController
Navigation Item (Root View Controller)	UINavigationItem
Detail View Controller	DetailViewController

SplitView.xcodeproj



Outlets

detailViewController	Detail View Controller
rootViewController	Root View Controll...
splitViewController	Split View Controller
window	Window

Referencing Outlets

delegate	File's Owner
New Referencing Outlet	

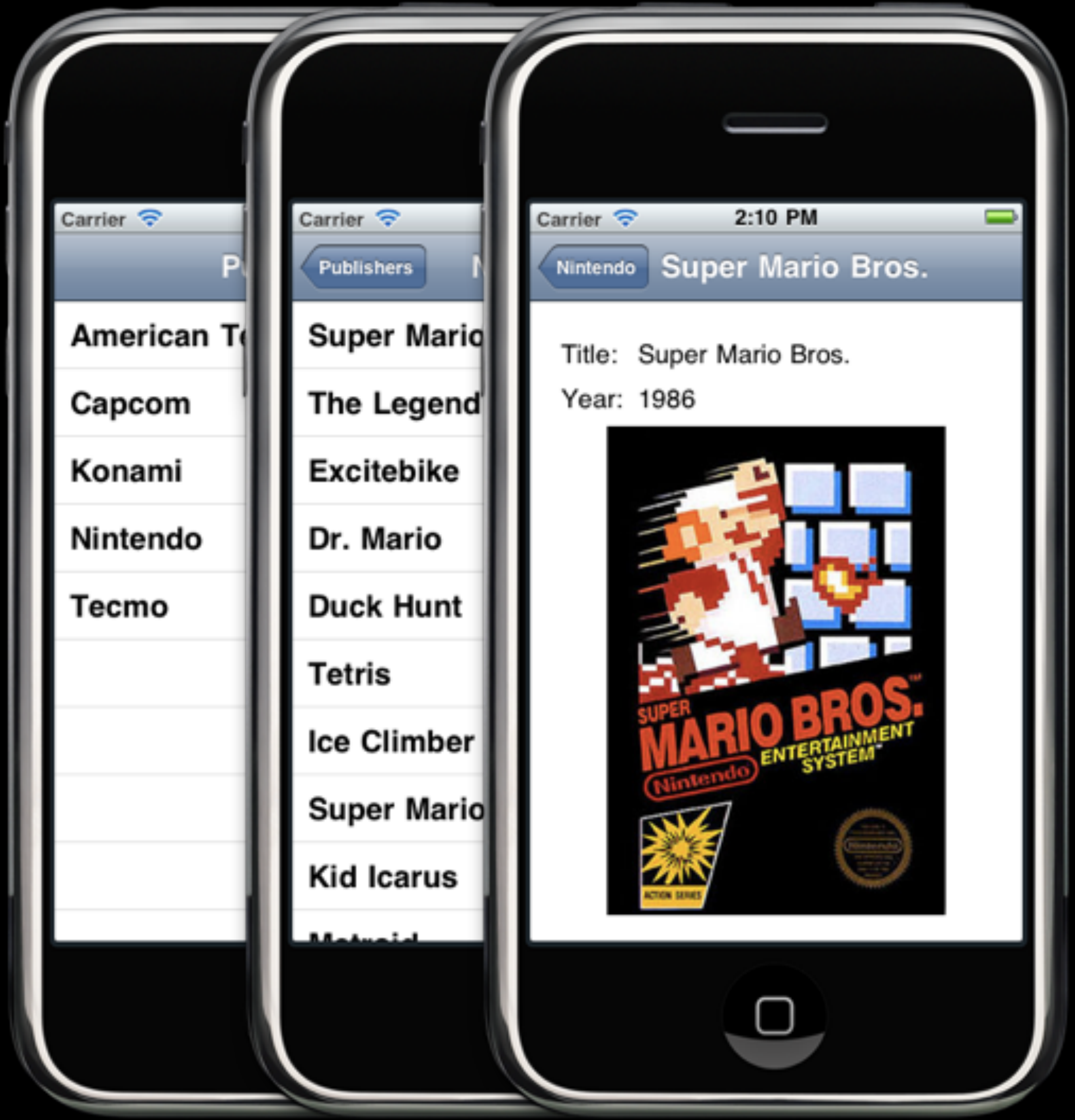
Notes About Split-view Based Project Layout

- RootViewController has an property/outlet for the detailViewController, so it can be manipulated from anywhere within the RootViewController class
  - Including the -tableView:didSelectRowAtIndexPath: method for when an item in list is tapped
- DetailViewController implements both popover and split-view delegate methods to handle the showing and hiding of the button (which brings up the popover)

Split View Example

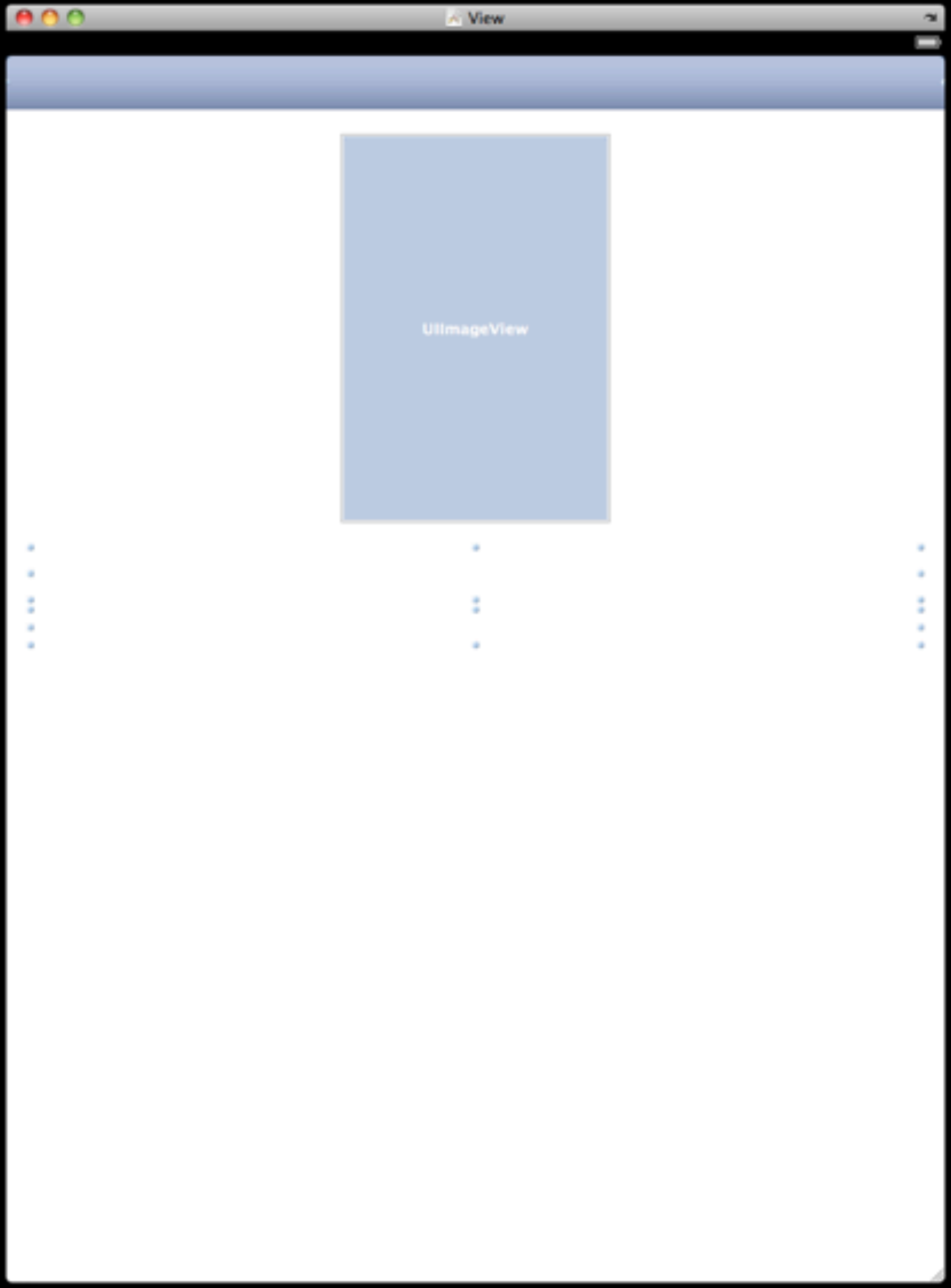
Video Game Example

- Remember the video game example from the Navigation and Tab Bar Apps lecture?
- Let's recreate this app for iPad using a Split-view based app



DetailViewController.xib

- Pretty basic — contains the following...
  - UIImageView for cartridge artwork
  - UILabels for publisher and year published
- All 3 wired up as outlets



DetailViewController.h

```
#import <UIKit/UIKit.h>

@interface DetailViewController : UIViewController
                     <UIPopoverControllerDelegate, UISplitViewControllerDelegate> {

    UIPopoverController *popoverController;
    id detailItem;
    UIToolbar *toolbar;
}

@property (nonatomic, retain) IBOutlet UIToolbar *toolbar;
@property (nonatomic, retain) id detailItem;

@property (nonatomic, retain) IBOutlet UIImageView *gameImage;
@property (nonatomic, retain) IBOutlet UILabel *gameTitle;
@property (nonatomic, retain) IBOutlet UILabel *gameYear;

@end
```

DetailViewController.m

```
#import "DetailViewController.h"
#import "RootViewController.h"
#import "VideoGame.h"

@interface DetailViewController ()
@property (nonatomic, retain) UIPopoverController *popoverController;
- (void)configureView;
@end

@implementation DetailViewController

@synthesize toolbar, popoverController, detailItem;
@synthesize gameImage, gameTitle, gameYear;

/* ... */
```

DetailViewController.m

```
#pragma mark -
#pragma mark Managing the detail item

/*
 When setting the detail item, update the view and
 dismiss the popover controller if it's showing.
 */
- (void)setDetailItem:(id)newDetailItem {
    if (detailItem != newDetailItem) {
        [detailItem release];
        detailItem = [newDetailItem retain];

        // Update the view.
        [self configureView];
    }

    if (self.popoverController != nil) {
        [self.popoverController dismissPopoverAnimated:YES];
    }
}
```

DetailViewController.m

```
/* ... */

- (void)configureView {

  // Update the user interface for the detail item.
  VideoGame *game = (VideoGame *)detailItem;
  self.gameTitle.text = game.title;
  self.gameYear.text = [NSString stringWithFormat:@"%d", game.year];
  NSCharacterSet *nonAlphaNum = [[NSCharacterSet alphanumericCharacterSet]
                                 invertedSet];
  NSString *fileName = [[game.title
                        componentsSeparatedByCharactersInSet:nonAlphaNum]
                        componentsJoinedByString:@""];
  self.gameImage.image = [UIImage imageNamed: fileName];
}

/* ... */
```

DetailViewController.m

```
/* ... */
#pragma mark -
#pragma mark Split view support

- (void)splitViewController: (UISplitViewController*)svc willHideViewController:
(UIViewController *)aViewController withBarButtonItem:(UIBarButtonItem*)
barButtonItem forPopoverController: (UIPopoverController*)pc {
    barButtonItem.title = @"Root List";
    NSMutableArray *items = [[toolbar items] mutableCopy];
    [items insertObject:barButtonItem atIndex:0];
    [toolbar setItems:items animated:YES];
    [items release];
    self.popoverController = pc;
}

- (void)splitViewController: (UISplitViewController*)svc willShowViewController:
(UIViewController *)aViewController invalidatingBarButtonItem:(UIBarButtonItem *)
barButtonItem {
    NSMutableArray *items = [[toolbar items] mutableCopy];
    [items removeObjectAtIndex:0];
    [toolbar setItems:items animated:YES];
    [items release];
    self.popoverController = nil;
}
```

DetailViewController.m

```
/* ... */

#pragma mark -
#pragma mark Rotation support

// Ensure that the view controller supports rotation and that
// the split view can therefore show in both portrait and landscape.
- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)
                                                              interfaceOrientation {
    return YES;
}

/* ... */

@end
```

RootViewController.h

```
#import <UIKit/UIKit.h>

@class DetailViewController;

@interface RootViewController : UITableViewController {
    DetailViewController *detailViewController;
    NSMutableArray *games;
}

@property (nonatomic, retain) IBOutlet DetailViewController *detailViewController;


@end
```

RootViewController.m

```
#import "RootViewController.h"
#import "DetailViewController.h"
#import "VideoGame.h"

@implementation RootViewController

@synthesize detailViewController;
#pragma mark -
#pragma mark View lifecycle

- (void)viewDidLoad {
    [super viewDidLoad];
    self.clearsSelectionOnViewWillAppear = NO;
    self.contentSizeForViewInPopover = CGSizeMake(320.0, 600.0);
    games = [[NSMutableArray alloc] initWithObjects:
           [VideoGame videoGameWithTitle:@"Super Mario Bros." year:1986],
           [VideoGame videoGameWithTitle:@"The Legend of Zelda" year:1986],
           /* ... many VideoGames clipped ... */
           [VideoGame videoGameWithTitle:@"Mega Man 2" year:1989],
           [VideoGame videoGameWithTitle:@"River City Ransom" year:1990],
           nil
           ];
}
/* ... */
```

RootViewController.m

```
/* ... */

// Ensure that the view controller supports rotation and that
// the split view can therefore show in both portrait and landscape.
- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)
                                                              interfaceOrientation {
  return YES;
}

#pragma mark -
#pragma mark Table view data source

- (NSInteger)numberOfSectionsInTableView:(UITableView *)aTableView {
    // Return the number of sections.
    return 1;
}


- (NSInteger)tableView:(UITableView *)aTableView
 numberOfRowsInSection:(NSInteger)section {
    // Return the number of rows in the section.
    return [games count];
}

/* ... */
```

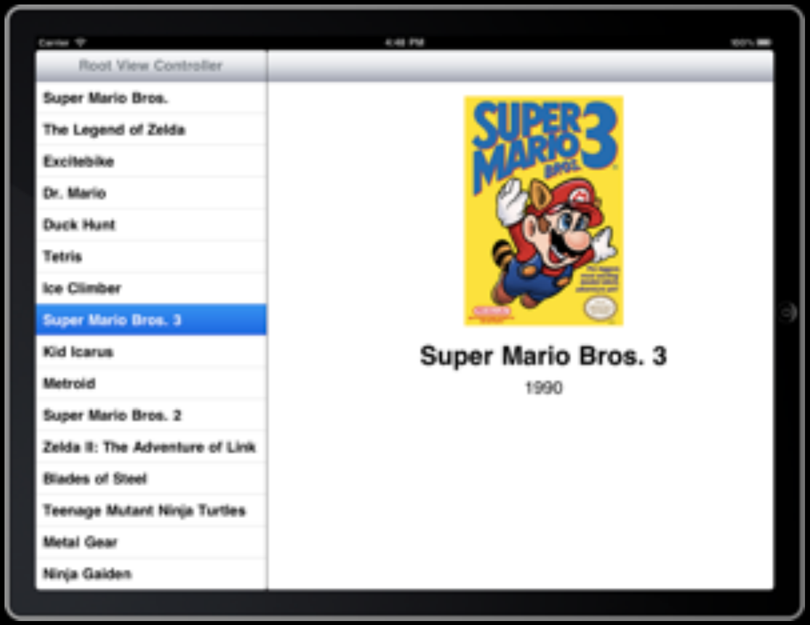
RootViewController.m

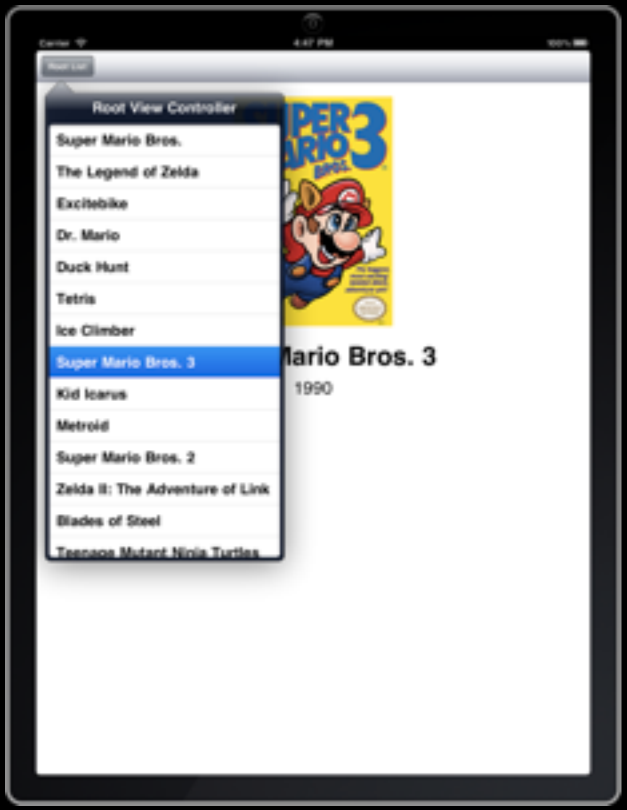
```
/* ... */

- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:
                                                         (NSIndexPath *)indexPath {

  static NSString *CellIdentifier = @"CellIdentifier";

  // Dequeue or create a cell of the appropriate type.
  UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier];
  if (cell == nil) {
    cell = [[[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault
                                   reuseIdentifier:CellIdentifier] autorelease];
    cell.accessoryType = UITableViewCellAccessoryNone;
  }

  // Configure the cell.
  cell.textLabel.text = [[games objectAtIndex:indexPath.row] title];
  return cell;
}

/* ... */
```

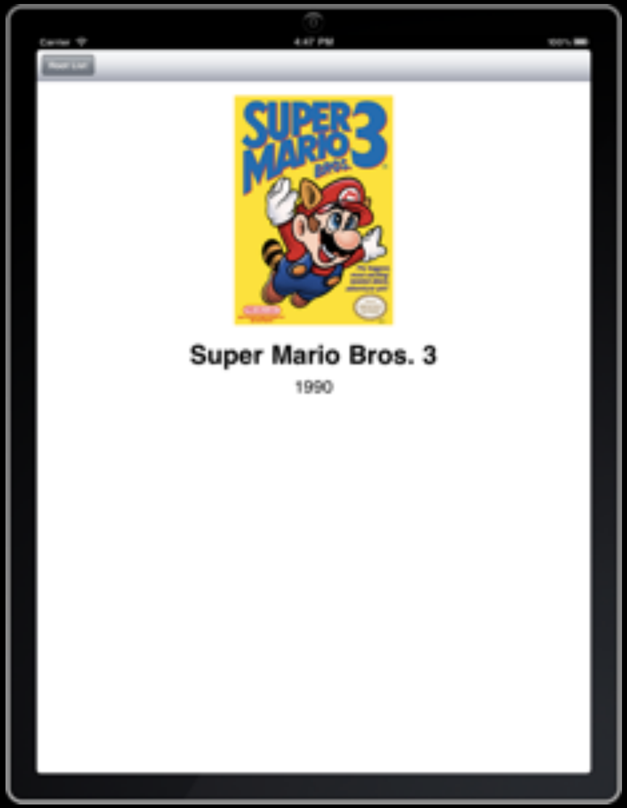
RootViewController.m

```
/* ... */

#pragma mark -
#pragma mark Table view delegate

- (void)tableView:(UITableView *)aTableView
                                  didSelectRowAtIndexPath:(NSIndexPath *)indexPath {

  /*
    When a row is selected, set the detail view controller's detail item
    to the item associated with the selected row.
  */
  detailViewController.detailItem = [games objectAtIndex: indexPath.row];
}

/* ... */

@end
```

The Resulting App







Universal Apps

iOS Universal Apps

- You can create a “universal” app that runs on both iPad and non-iPad devices
- This allows the owner of the app to install and run the same app on any iOS device

Universal or Not?

- Universal App
  - If minor (mostly UI) changes, you could easily separate out the different UIs and leverage (mostly) the same backend code
- Generate 2 targets
  - If your app shares a fair amount of code but has different logic, feel, etc., then you may want to set the project up to generate to targes
- Different projects
  - If there's not that much in common

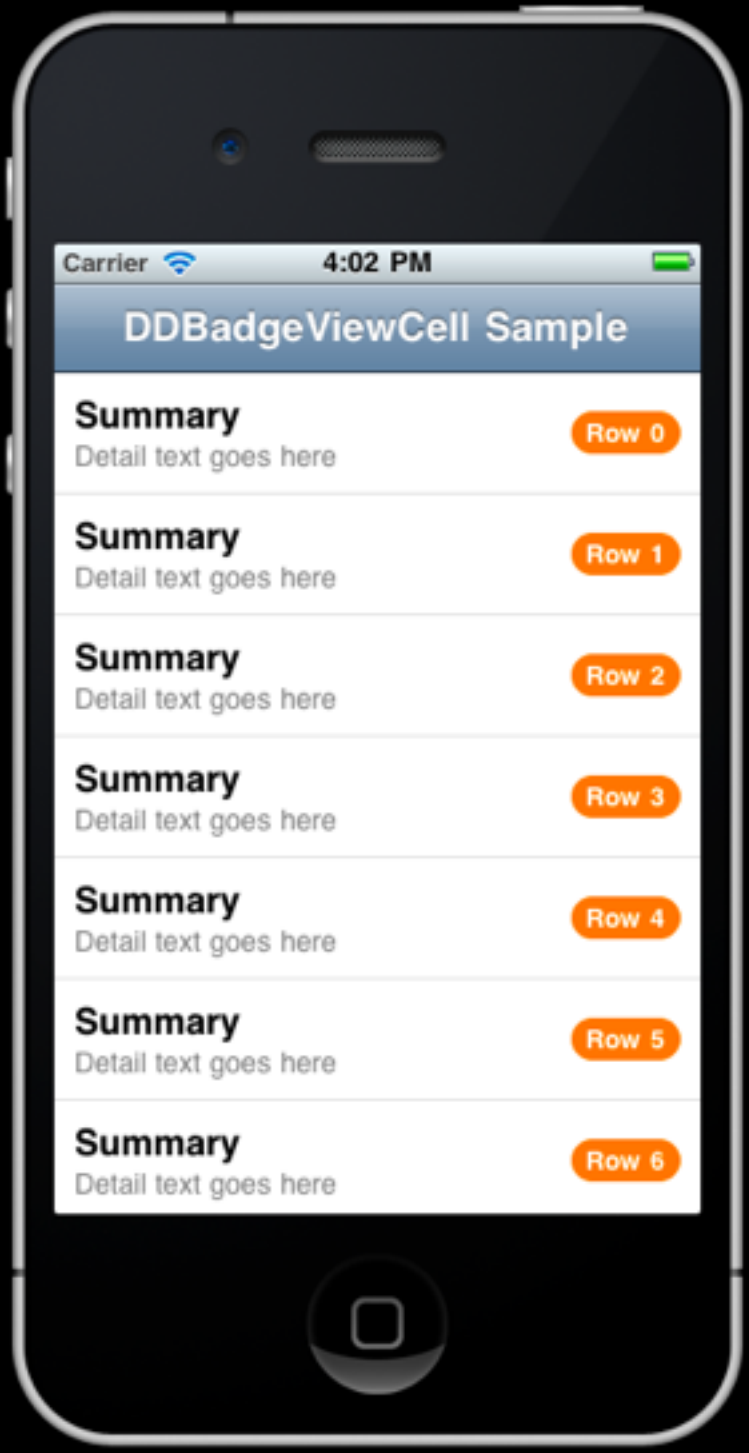
Universal App Example

Example

- Let’s create a basic app that displays some data...
  - On the iPhone or iPod touch we’ll only display the data in tabular form
  - On the iPad, we’ll take advantage of the larger display and also show a graph representation of the data
- We’ll start this project as a iPhone View-based app, then add iPad support
  - I’ve named the app “Universal” in the example that follows

DDBadgeViewCell

- For this example, we're going to utilize a custom cell renderer from the DDBadgeViewCell library
- For more info see...
  - https://github.com/digdog/DDBadgeViewCell



Summary
Detail text goes here
Row 0

Summary
Detail text goes here
Row 1

Summary
Detail text goes here
Row 2

Summary
Detail text goes here
Row 3

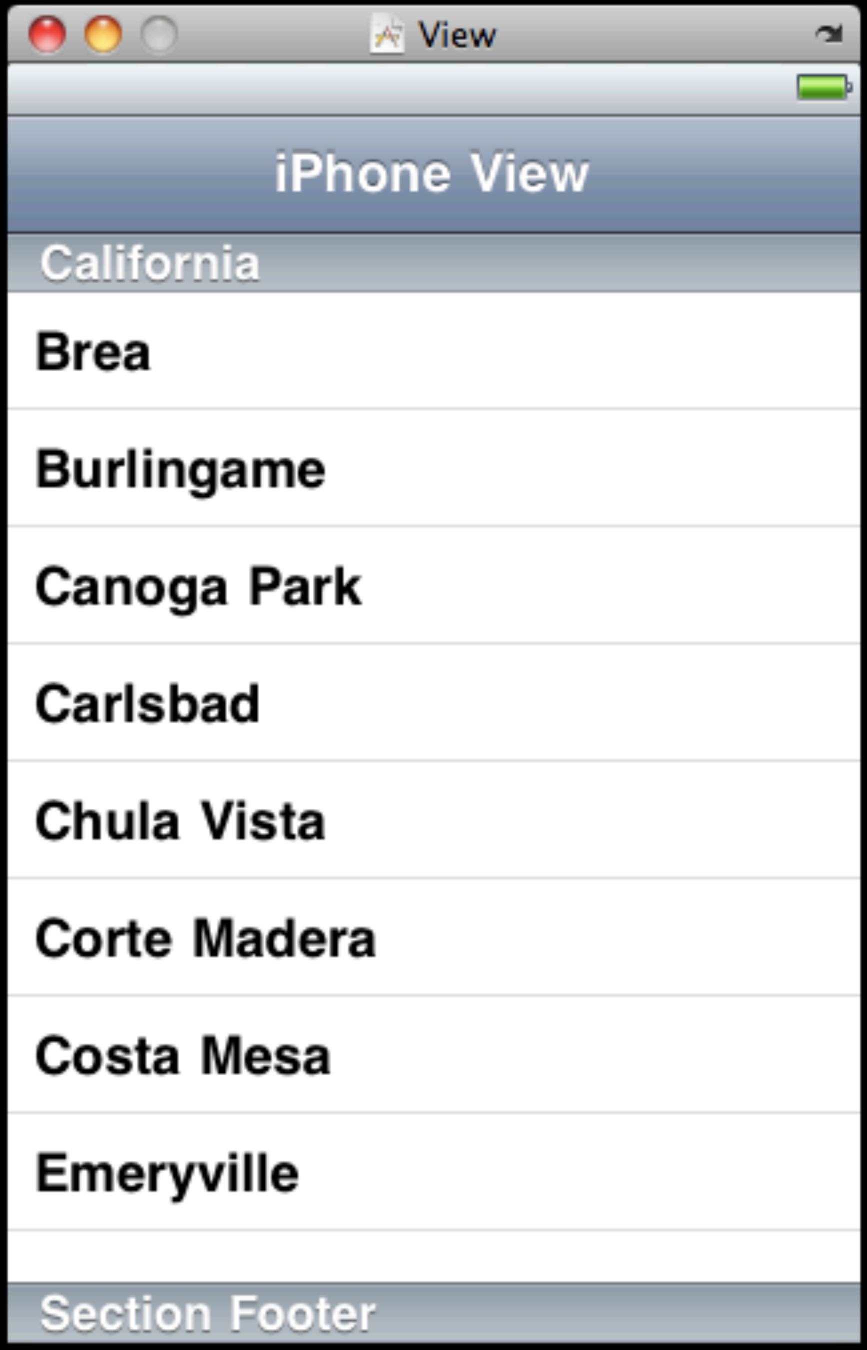
Summary
Detail text goes here
Row 4

Summary
Detail text goes here
Row 5

Summary
Detail text goes here
Row 6

UniversalViewController.xib

- Here we'll add...
  - A basic nav bar with title
  - A table view which takes up the rest of the screen
- Our view controller will implement the table data source and delegate methods
  - So, we'll need to wire these up to File's Owner (the view controller)



UniversalViewController.h

```
#import <UIKit/UIKit.h>

@interface UniversalViewController : UIViewController
          <UITableViewDelegate, UITableViewDataSource> {

    NSArray *data;
    NSArray *days;

}

@end
```

UniversalViewController.m

```
#import "UniversalViewController.h"
#import "DDBadgeViewCell.h"

@implementation UniversalViewController

#pragma mark -
#pragma mark Table Data Source & Delegate Methods

- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
    return 1;
}

- (NSInteger)tableView:(UITableView *)tableView
 numberOfRowsInSection:(NSInteger)section {
    return [data count];
}

/* ... */
```

UniversalViewController.m

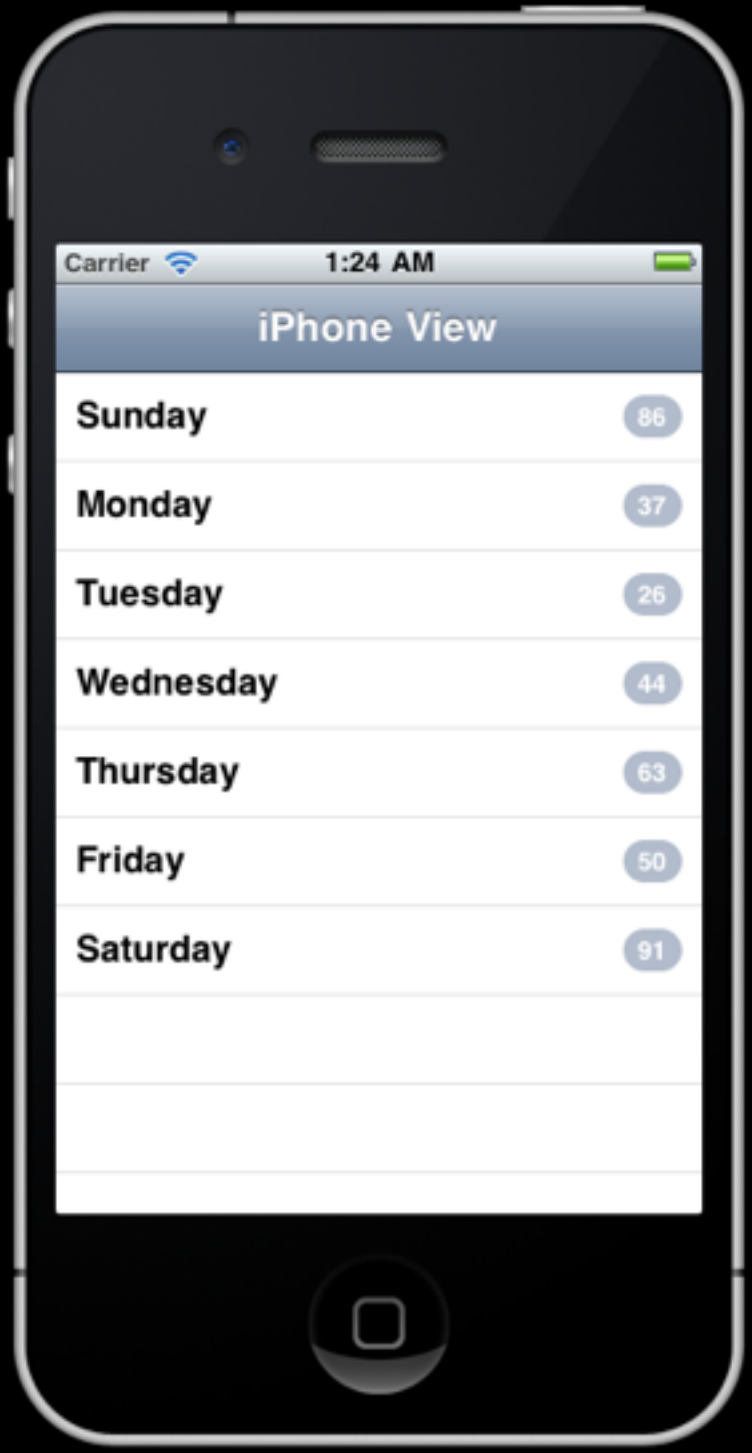
```
/* ... */

- (UITableViewCell *)tableView:(UITableView *)tableView
         cellForRowAtIndexPath:(NSIndexPath *)indexPath {

  static NSString *CellIdentifier = @"Cell";

  DDBadgeViewCell *cell = (DDBadgeViewCell *)[tableView
                                dequeueReusableCellWithIdentifier:CellIdentifier];
  if (cell == nil) {
    cell = [[[DDBadgeViewCell alloc] initWithStyle:UITableViewCellStyleDefault
                                  reuseIdentifier:CellIdentifier] autorelease];
  }

  // Configure the cell
  cell.summary = [[days objectAtIndex:indexPath.row] description];
  cell.badgeText = [[data objectAtIndex:indexPath.row] description];
  cell.badgeColor = [UIColor colorWithRed:176/255.0 green:188/255.0
                         blue:205/255.0 alpha:1.0];

  return cell;
}

/* ... */
```

UniversalViewController.m

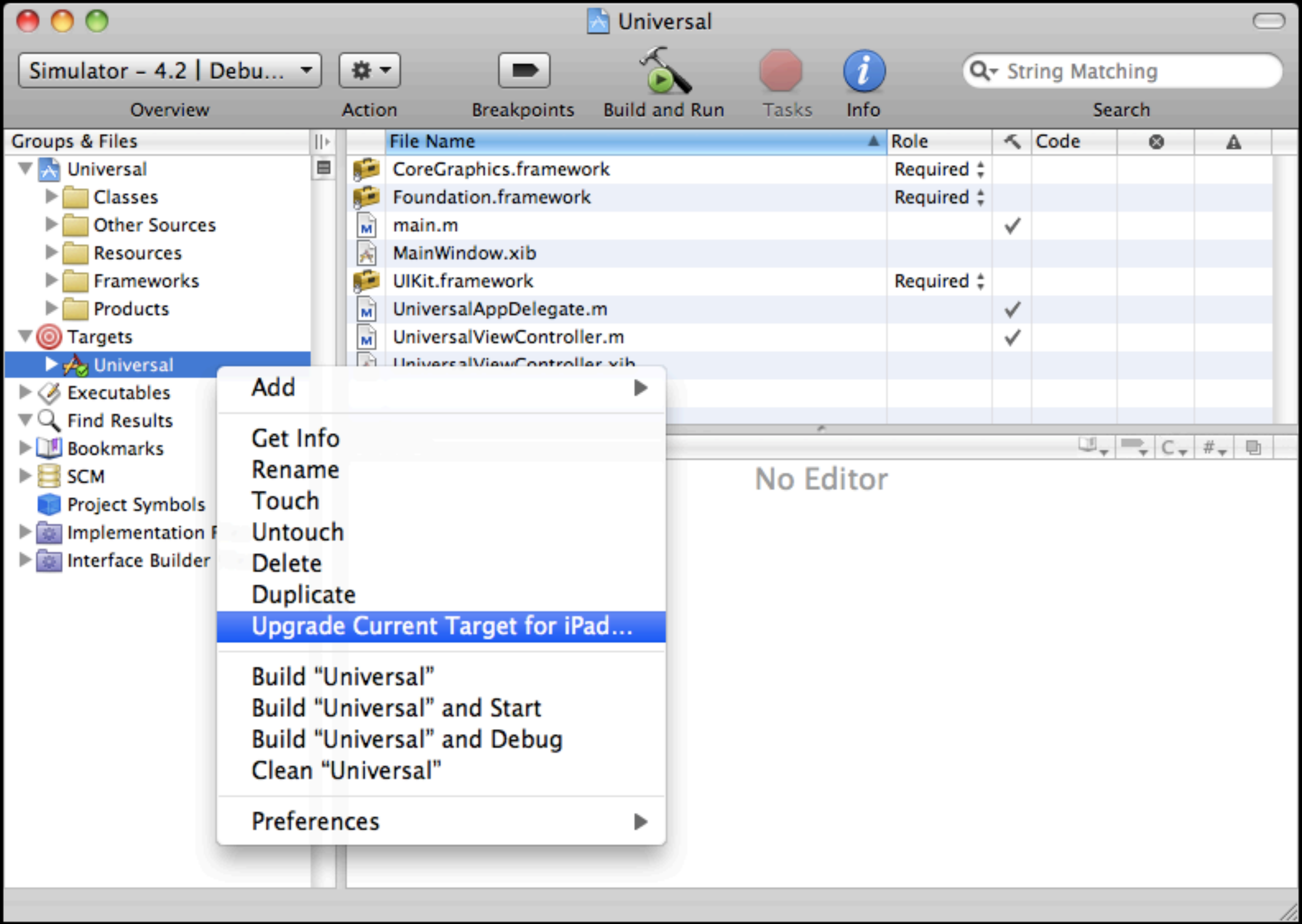
```
/* ... */

- (void)viewDidLoad {
  [super viewDidLoad];
  data = [[NSArray alloc] initWithObjects:
          [NSNumber numberWithInt:86],
          [NSNumber numberWithInt:37],
          [NSNumber numberWithInt:26],
          [NSNumber numberWithInt:44],
          [NSNumber numberWithInt:63],
          [NSNumber numberWithInt:50],
          [NSNumber numberWithInt:91],
          nil];
  days = [[NSArray alloc] initWithObjects:
          @"Sunday",
          @"Monday",
          @"Tuesday",
          @"Wednesday",
          @"Thursday",
          @"Friday",
          @"Saturday",
          nil];
}

/* ... */
@end
```

The Resulting App Run on an iPhone

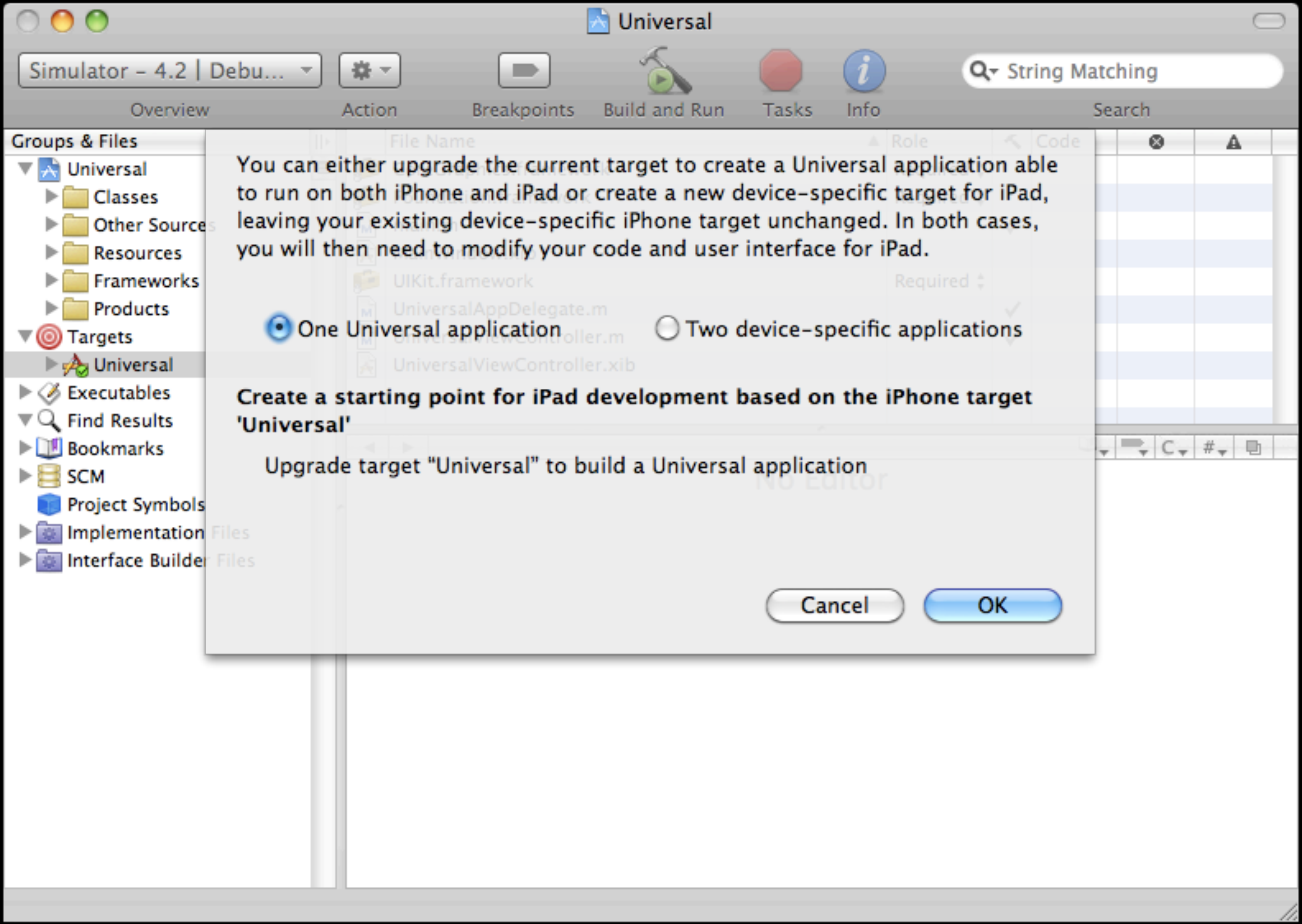
- Here's what the app looks like thus far when we build and run it
- Let's add iPad support to the app...



Adding iPad Support



Adding iPad Support



You can either upgrade the current target to create a Universal application able to run on both iPhone and iPad or create a new device-specific target for iPad, leaving your existing device-specific iPhone target unchanged. In both cases, you will then need to modify your code and user interface for iPad.

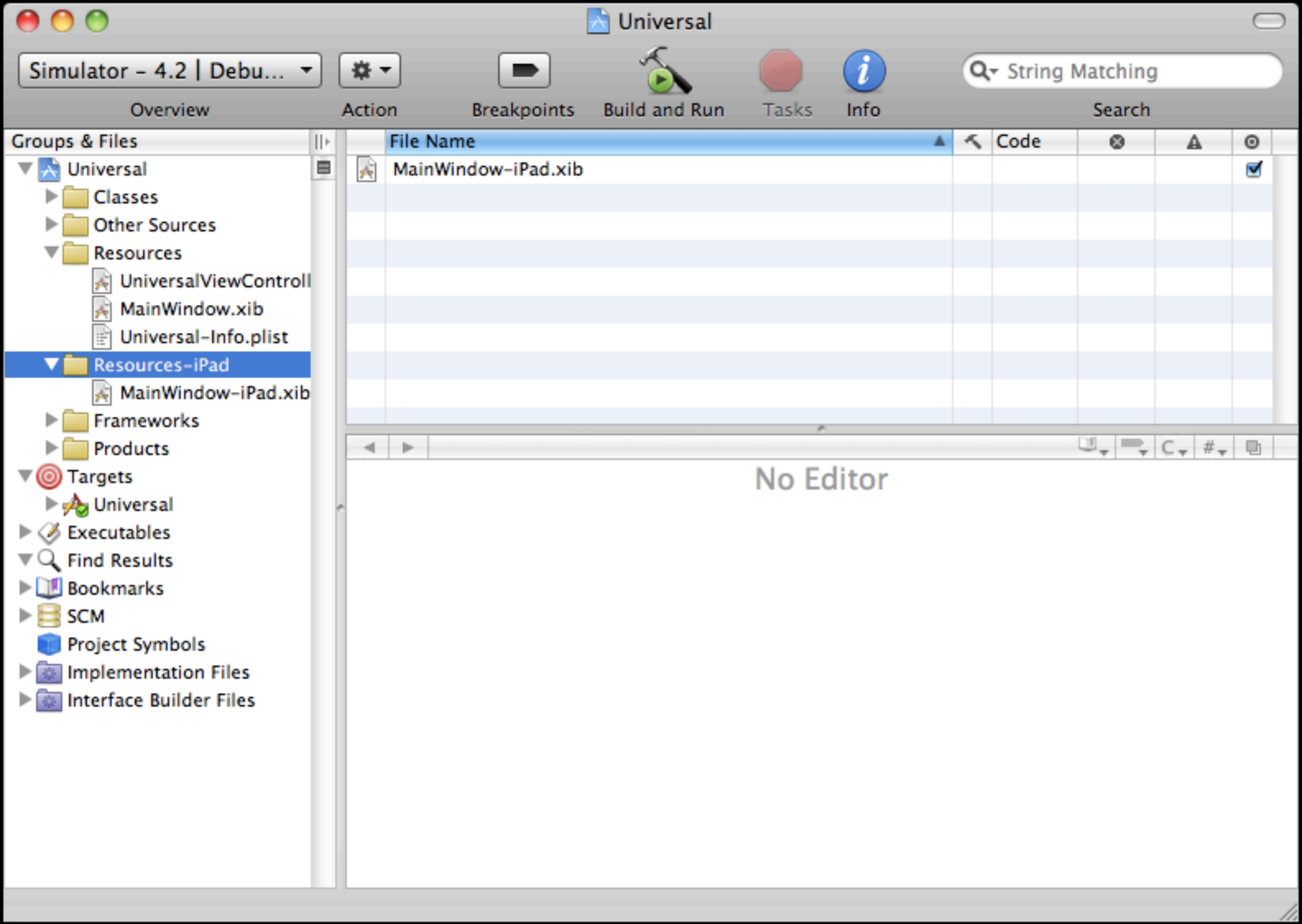
[✓ One Universal application] Two device-specific applications

Create a starting point for iPad development based on the iPhone target 'Universal'

Upgrade target "Universal" to build a Universal application

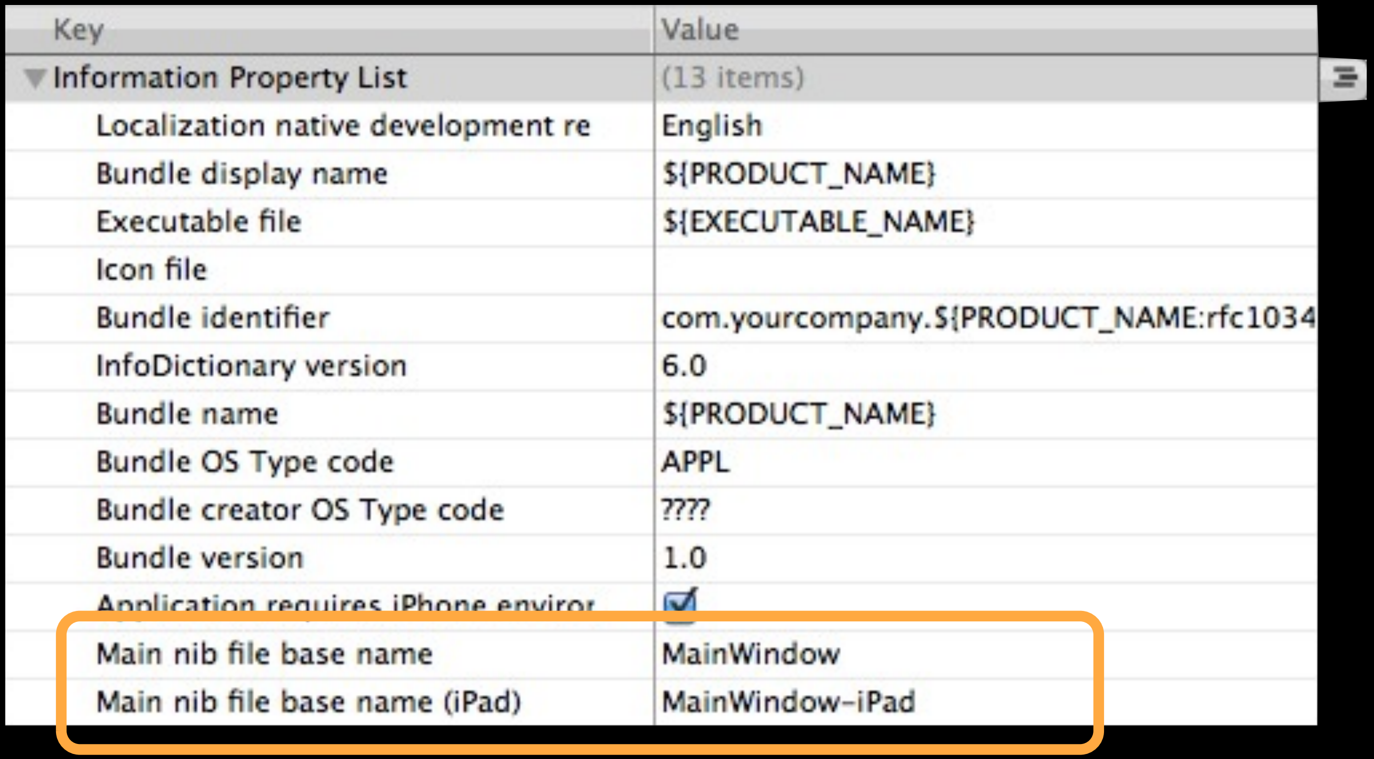
Cancel OK

Adding iPad Support



Info Plist

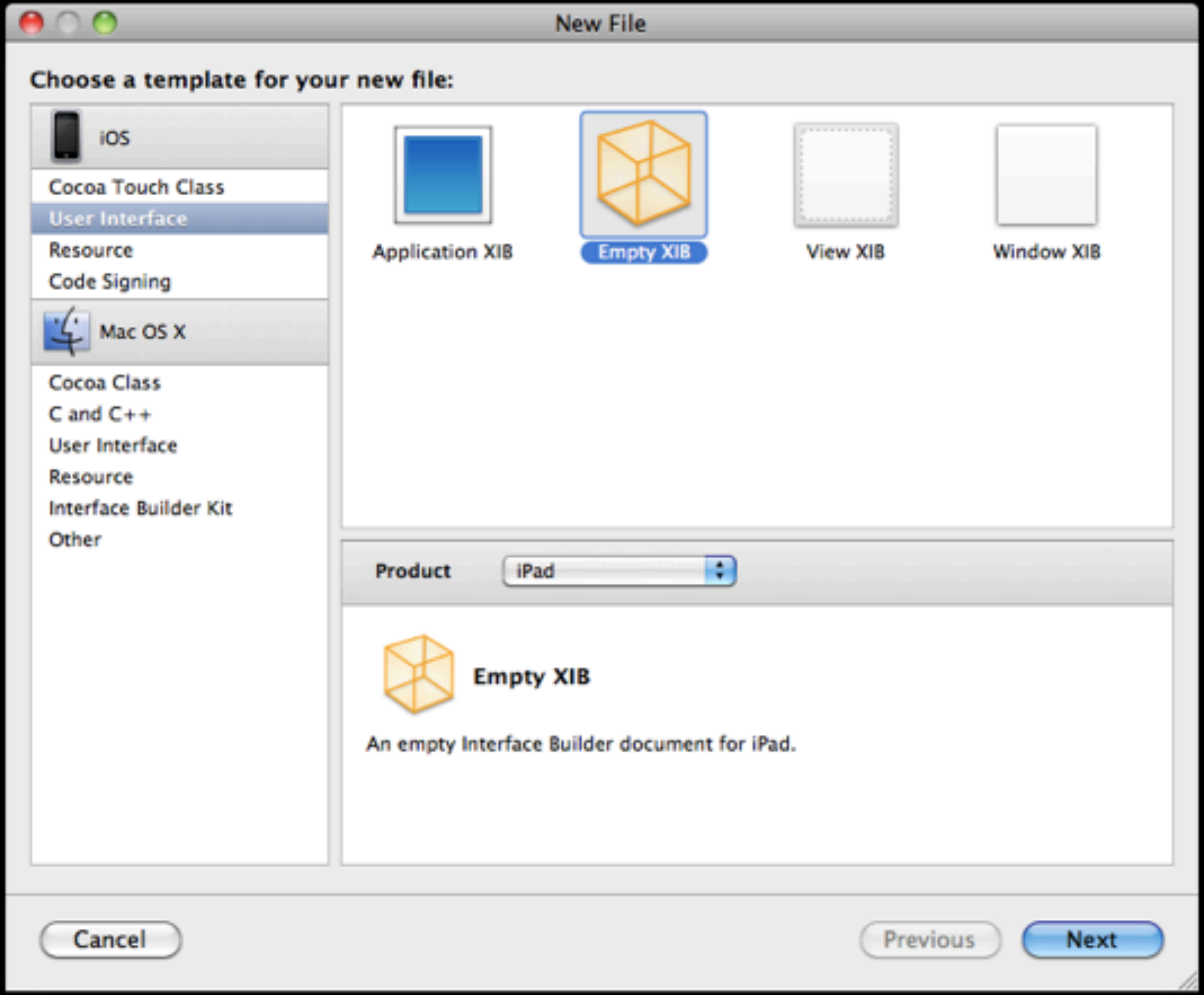
- Once we perform this operation, if you open the app's info.plist file you'll notice different main NIB keys for both iPhone and iPad...



Key	Value
Information Property List	(13 items)
Localization native development re	English
Bundle display name	${PRODUCT_NAME}
Executable file	${EXECUTABLE_NAME}
Icon file	
Bundle identifier	com.yourcompany.${PRODUCT_NAME:rfc1034
InfoDictionary version	6.0
Bundle name	${PRODUCT_NAME}
Bundle OS Type code	APPL
Bundle creator OS Type code	????
Bundle version	1.0
Application requires iPhone enviror	✓
Main nib file base name	MainWindow
Main nib file base name (iPad)	MainWindow-iPad

Adding a Custom iPad View

- Create a new NIB for the iPad to use instead of the default iPhone NIB — let's call it UniversalViewController-iPad.xib



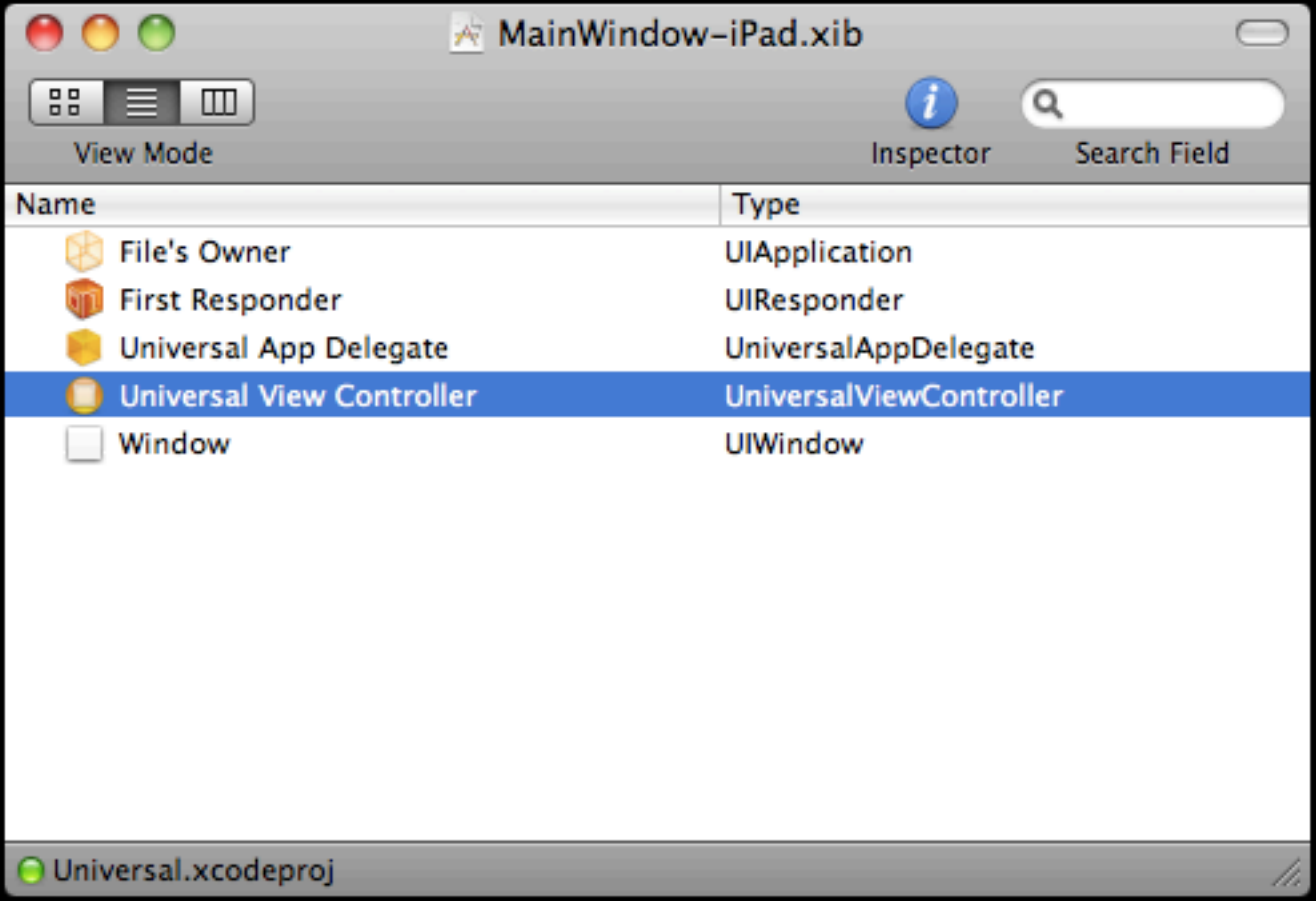
Product: iPad

Empty XIB

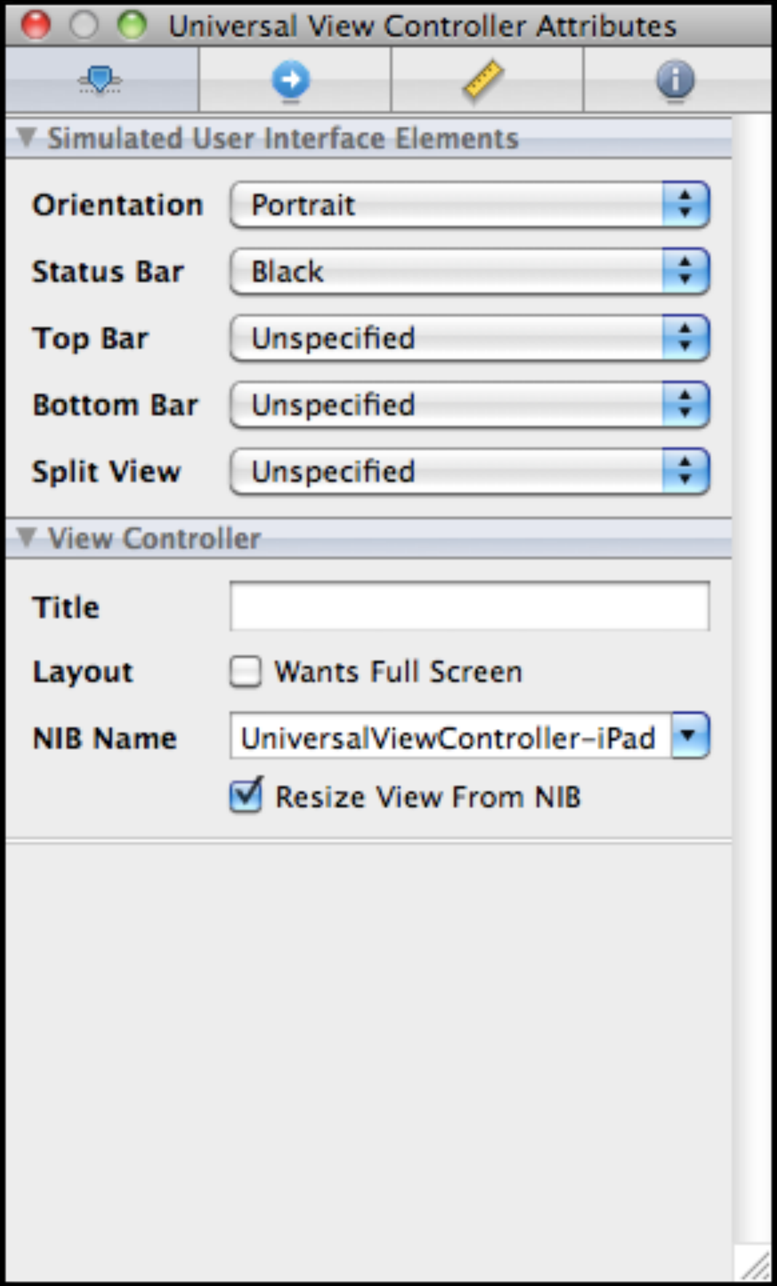
An empty Interface Builder document for iPad.

MainWindow-iPad.xib

- Open the inspector on the View Controller and change it to the newly created UniveralViewController-iPad



Name	Type
File's Owner	UIApplication
First Responder	UIResponder
Universal App Delegate	UniversalAppDelegate
Universal View Controller	UniversalViewController
Window	UIWindow



Simulated User Interface Elements

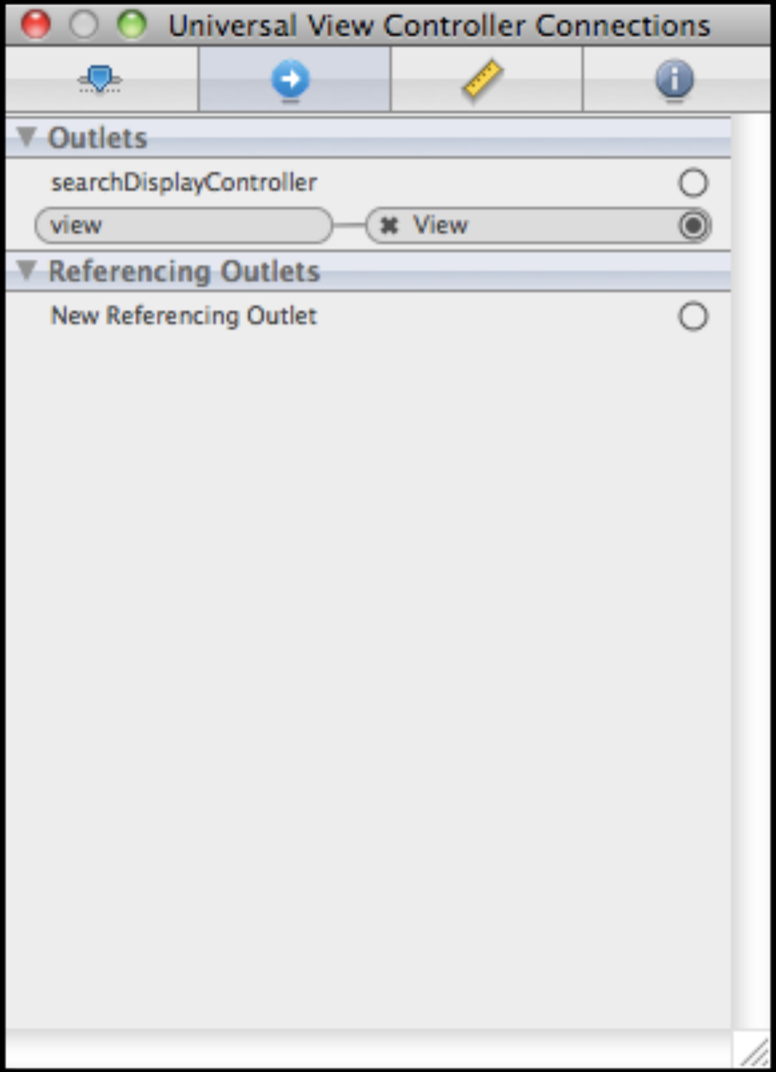
Orientation	Portrait
Status Bar	Black
Top Bar	Unspecified
Bottom Bar	Unspecified
Split View	Unspecified

View Controller

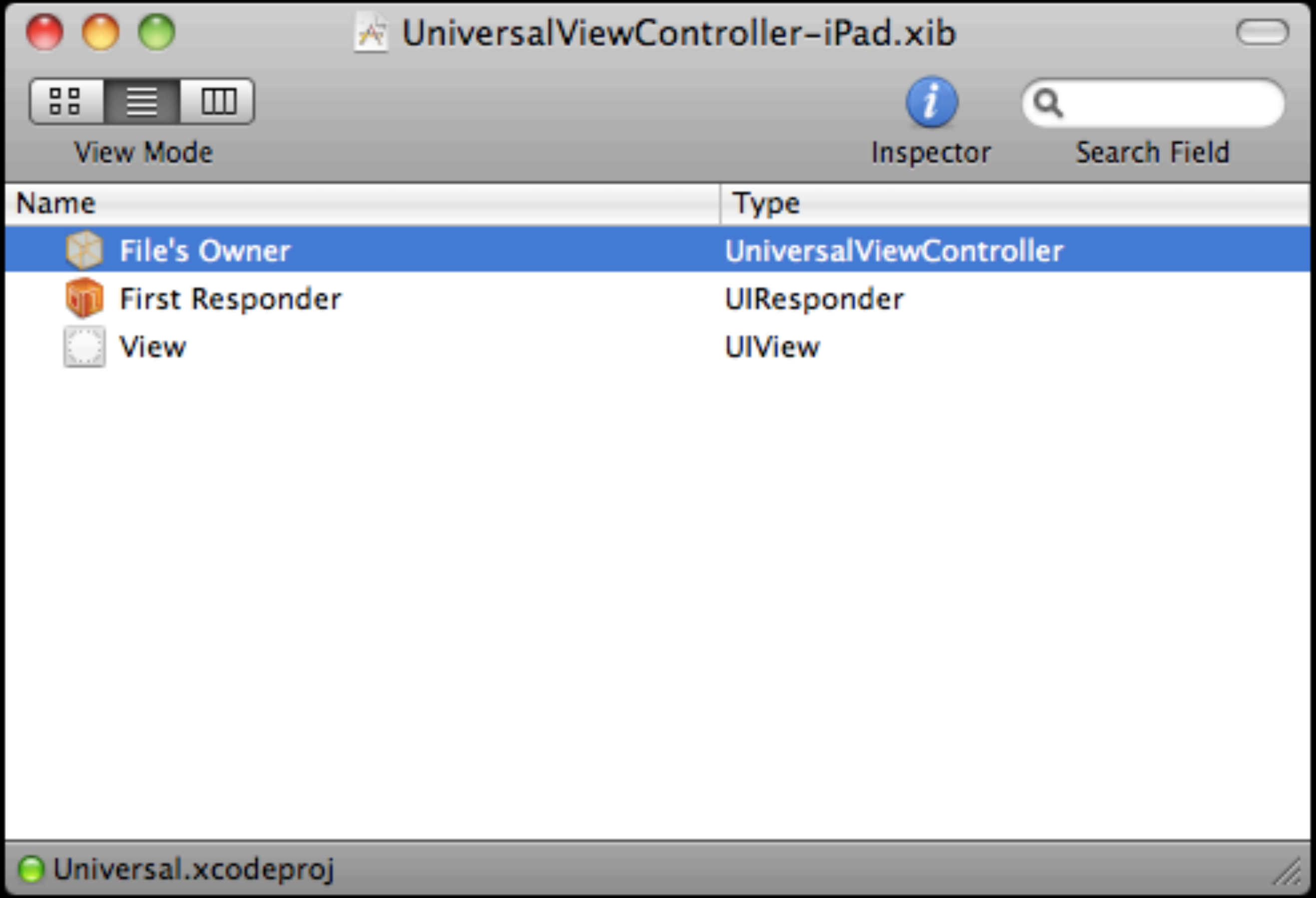
Title	
Layout	☐ Wants Full Screen
NIB Name	UniversalViewController-iPad
	☑ Resize View From NIB

UniversalViewController-iPad.xib

- Change the class on File's Owner to UniversalViewController
- Drop a UIView into the NIB from the library and wire it as File's Owner's view



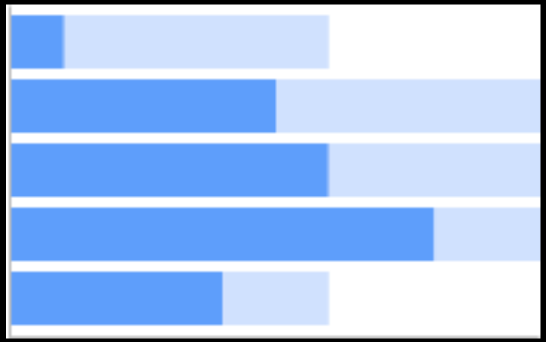
searchDisplayController	
view	View

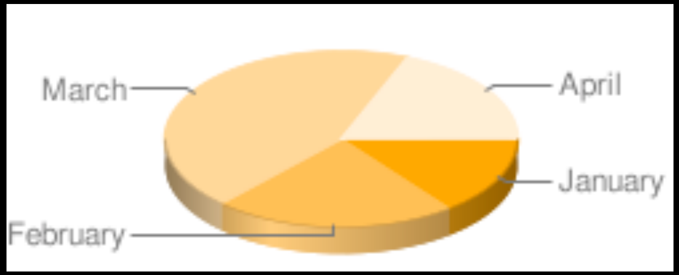


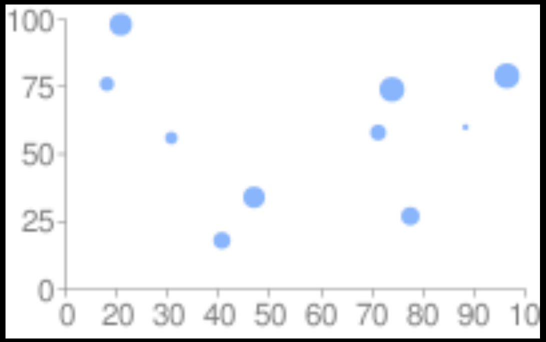
Name	Type
File's Owner	UniversalViewController
First Responder	UIResponder
View	UIView

Google Charts API

- For the extended iPad version, we're also going to render a chart corresponding to the data
- To do so, we're going leverage the Google Charts API
- Basically request a URL encoding the data into the request
- For more info see...
  - http://code.google.com/apis/chart/

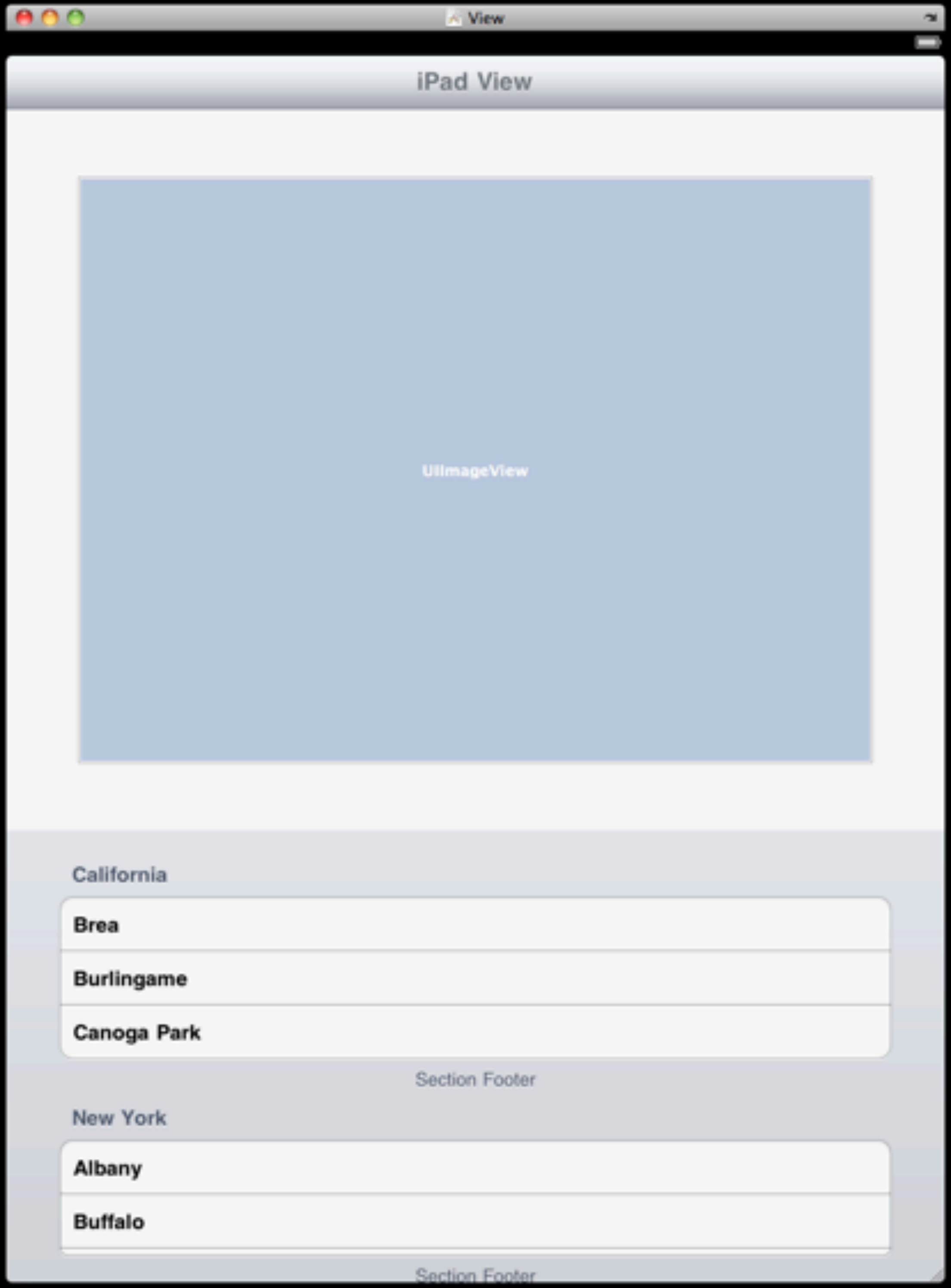






UniversalViewController-iPad.xib

- In the iPad specific NIB, we'll go ahead and place a table view and wire it up to File's Owner (the view controller)
- We'll also add a UIImage as an outlet to our view controller and wire it up here



UniversalViewController.h

```
#import <UIKit/UIKit.h>

@interface UniversalViewController : UIViewController
          <UITableViewDelegate, UITableViewDataSource> {

    NSArray *data;
    NSArray *days;

}

@property(nonatomic, retain) IBOutlet UIImageView *chart;

@end
```

Added outlet

UniversalViewController.m

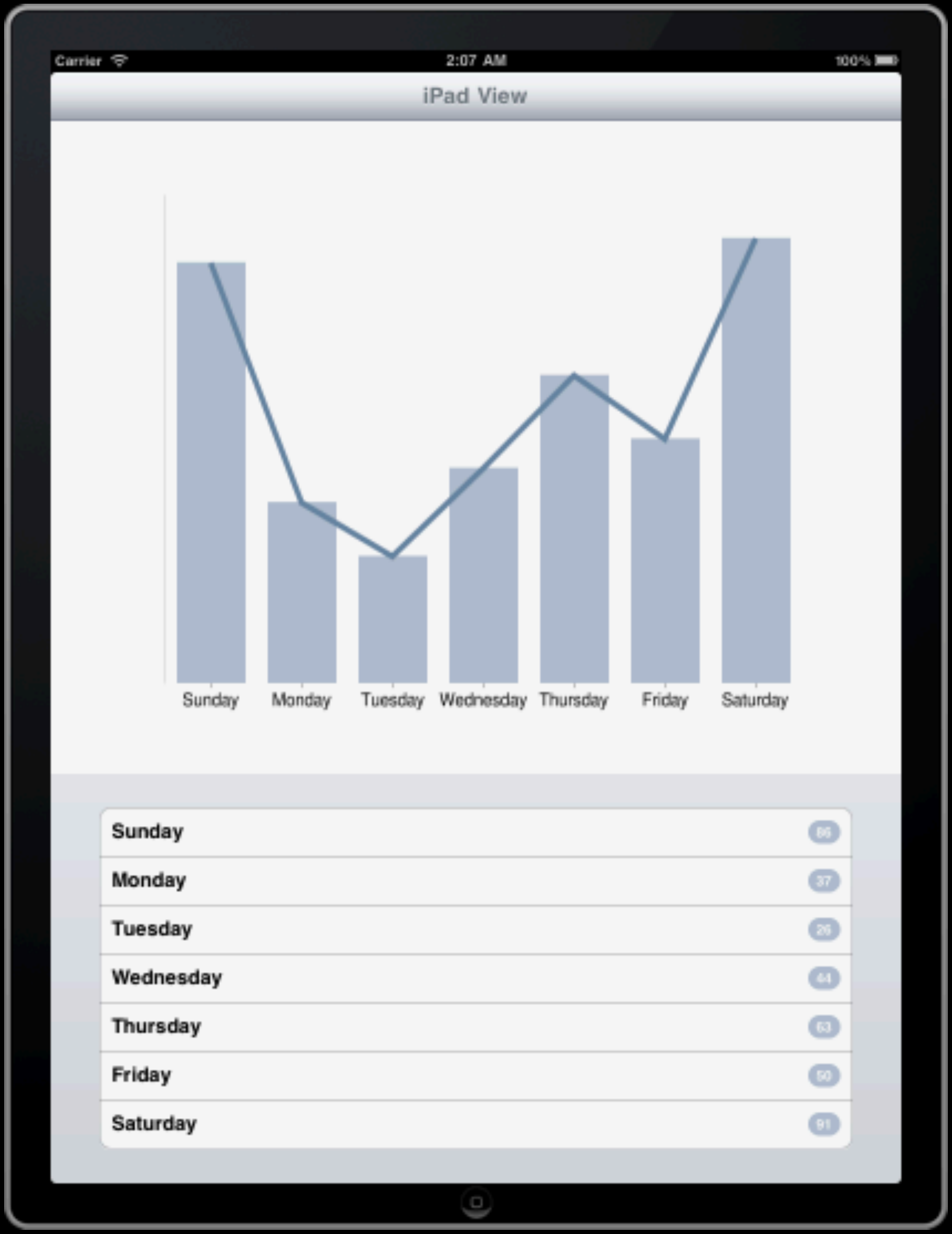
```
#import "UniversalViewController.h"
#import "DDBadgeViewCell.h"

@implementation UniversalViewController

@synthesize chart;

/* ... everything same as before ... */

- (void)viewDidLoad {

  /* ... same viewDidLoad body as before ... */

  /* ... but, we're getting ready to add some more code to this method ... */

  /* ... */
```

Added synthesize statement

UniversalViewController.m

```
  /* ... */

  /* run this code only if on iPad */
  if (UI_USER_INTERFACE_IDIOM() == UIUserInterfaceIdiomPad) {

    /* build up query string */
    NSString *chartStr = [NSString stringWithFormat:@"http://chart.apis.google.com/
chart?chf=bg,s,F7F7F7&chxl=0:|%@&chxr=0,0,103.333&chxs=0,000000,15,0,t,
000000&chxt=x&chbh=a,5,20&chs=650x443&cht=bvg&chco=B0BCCD&chd=t:
%@&chma=44,0,10&chm=D,6D84A2,0,0,5,1",
                              [days componentsJoinedByString:@"|"],
                              [data componentsJoinedByString:@","]
                             ];
    chartStr = [chartStr stringByReplacingOccurrencesOfString:@"|"
                             withString:@"%7C"];

    /* fetch URL as data, create UIImage and set as view's image property */
    NSURL *chartURL = [NSURL URLWithString:chartStr];
    NSData *chartData = [NSData dataWithContentsOfURL:chartURL];
    UIImage *chartImage = [[[UIImage alloc] initWithData:chartData] autorelease];
    self.chart.image = chartImage;
  }
}

/* ... */

@end
```

The Resulting App Run on an iPad



Sunday	86
Monday	37
Tuesday	26
Wednesday	44
Thursday	63
Friday	50
Saturday	91

Additional Resources

- View Controller Programming Guide for iOS — iPad-Specific Controllers section...
  - http://developer.apple.com/library/ios/#featuredarticles/ViewControllerPGforiPhoneOS/iPadControllers/iPadControllers.html
- iOS Application Programming Guide — Creating a Universal Application section...
  - http://developer.apple.com/library/ios/#documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/BuildTimeConfiguration/BuildTimeConfiguration.html#//apple_ref/doc/uid/TP40007072-CH7-SW24

For Next Class

- Implementing Common Application Behaviors section of the iOS Application Programming Guide
    - http://developer.apple.com/library/ios/#documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/StandardBehaviors/StandardBehaviors.html
- iOS Human Interface Guidelines
    - http://developer.apple.com/library/ios/#documentation/UserExperience/Conceptual/MobileHIG/
- Internationalization Programming Topics
    - http://developer.apple.com/library/ios/#documentation/MacOSX/Conceptual/BPInternational/