Outline

This demo has 8 distinct steps divided into four sections:

1. Create a Facebook app from the Facebook developer portal
    - Learn how and where to create a new Facebook app
2. Modify an iPhone app to show a custom header in a table view
    - Learn how to use the Interface Builder
    - Learn how to wire up components in a XIB with an implementation file
    - Learn about TableViewController
    - Learn about accessing Facebook user data
3. Add a new view controller to a tabbed menu
    - Learn how to create a new view controller from scratch
    - Get the religion about creating organized projects
4. Use Core Location to power the Facebook Place View Controller

By the end of this demo, you will have the skills necessary to build your own iOS app and employ the Facebook SDK for iOS to make your app social.

What's Not in This Demo

Things that are out of scope for this demo include signing and deploying to the App Store, using or creating custom controls, managing differences in code when using deprecated APIs, and a whole host of other things that makes iOS development an awesome and worthwhile endeavor.

Resources that you may love:

- Sick of your app looking ugly? Go to www.appdesignvault.com and get some UI templates.
- Want slick controls like you see in your favorite apps? Go to www.cocoacontrols.com and find exactly what you're looking for.
- Need some help or additional tutorials to speed up your learning? Go to www.raywenderlich.com for the best iOS tutorials on the planet.

Above all: do not give up. You'll love this, I promise.

System Requirements

This demo requires the following:

1. Install Xcode 4.4 or later
2. Install the Facebook iOS SDK (developers.facebook.com)
3. Download and install the iOS Templates (github.com/CoolAssPuppy)

Initial Setup

If you're repeating this demo, then this is what you'll need to do to reset everything:

1. Delete the Scrumptious Facebook App from developers.facebook.com
2. Delete the Scrumptious App from ~/Projects/Demos
3. Make sure the Facebook iOS SDK is installed in ~/Projects/Libraries
4. Reset the iOS Simulator
5. Add Facebook credentials to the iOS Simulator
6. Install the latest templates

Part 1: Create a new Facebook app

1. In the browser, visit developers.facebook.com and click on "Apps" in the menu
2. Click on "Create a New App"
3. Click Edit Settings for the Facebook App
4. Check off "Native iOS App" and create a Bundle Identifier
5. Make sure the "Login" radio button is enabled
6. Take note of the Facebook APP ID and the Bundle Identifier

Part 2: Create a new iPhone app

7. In Xcode, use the Basic template to create a new Facebook App
8. Add the Facebook SDK and the Resources module
9. Run it, show the Login with the Facebook mobile app, cancel out of it
10. Explain Facebook iOS 6.0 integration
11. Setup iOS 6.0 credentials in the simulator

12. Re-run the application and login

Part 3: Wire up the header

13. Add an outlet and function for the header in ProfileViewController.h:

```
@interface ProfileViewController : UITableViewController
<UITableViewDataSource, UITableViewDelegate>
{
    IBOutlet UIView *headerView;
}
-(UIView *)headerView;
```

14. Now implement the function in ProfileViewController.m:

```
// implement the header view
-(UIView *)headerView
{
    // if we haven't loaded the header view yet...
    if(!headerView) {
        // load the header view XIB
        [[NSBundle mainBundle] loadNibNamed:@"ProfileViewHeader"
owner:self options:nil];
    }

    return headerView;
}
```

Part 4: Add a header to the Profile view controller

15. Right-click on the Profile group and select “New File…”

16. Click on User Interface under iOS in the left nav, then select Empty

17. Select iPhone as the device

18. Name it ProfileViewHeader

19. Click on File’s Owner and select ProfileViewController in the identity inspector

20. Drag and drop a UIView control to the surface

21. Resize it so that it’s 100px high (you may need to select Size as “Freeform” in the identity inspector)

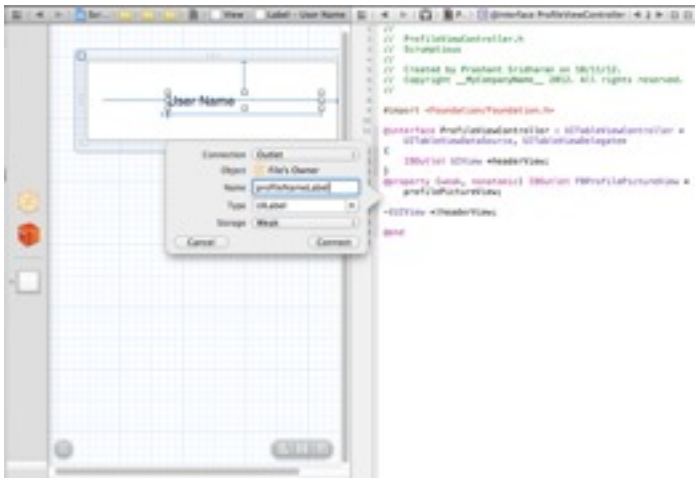
22. Drag and drop a UI View and a Label to the header view surface

23. Change the class on UIView to FBProfilePictureView

24. OPTION-click to open ProfileViewController.h

Now that we've wired the user interface to the implementation file, let's wire up the controls inside our interface. We do this using Outlets (a way for an implementation file to get the value of a user interface element) and Actions (a way for a user interface element to tell the implementation file that something just happened).

25. CTRL-drag and drop the outlets for the view and label to the header file. Name them profilePictureView and profileNameLabel.



26. CTRL-drag and drop from the File's Owner cube to the UIView surface to make the view visible from within the ProfileViewController (select headerView in the gray menu)

27. Create the populateUserDetails function:

```
// populate the header view properties with data from Facebook
-(void)populateUserDetails
{
    [[FBRequest requestForMe] startWithCompletionHandler:
         ^(FBRequestConnection *connection, NSDictionary<FBGraphUser>
*user, NSError *error) {
        if(!error) {
            self.profilePictureView.profileID = user.id;
            self.profileNameLabel.text = user.name;
        }
    }];
}
```

28. Call populateUserDetails from the headerView method:

```
// implement the header view
-(UIView *)headerView
{
    // if we haven't loaded the header view yet...
    if(!headerView) {
        // load the header view XIB
        [[NSBundle mainBundle] loadNibNamed:@"ProfileViewHeader"
owner:self options:nil];
        [self populateUserDetails];
    }

    return headerView;
}
```

29. Now we need to implement the delegate for headers in Tables:

```
-(UIView *)tableView:(UITableView *)tableView viewForHeaderInSection:
(NSInteger)section
{
    if (section == 0)
        return [self headerView];
    else
        return nil;
}

-(CGFloat)tableView:(UITableView *)tableView heightForHeaderInSection:
(NSInteger)section
{
    if(section == 0)
        return 130;
    else
        return 30;
}
```

30. Run the application.

Part 5: Add a new View Controller to the tabbed menu

I'm a big believer in having a clean, organized project explorer. Clutter is not good for the soul or mind of a software developer. As such, I like to organize files as views, models, controls, and utilities. In this case, we want to add a new tab to our application, which means we will add a new group of files under the "views" folder.

31. Right-click on "views" in the project explorer and select "Add Group". Name the group "places".

32. Right-click on "places" in the project explorer and select "New File..."

33. Select “Cocoa Touch” in the left nav and select “Objective-C class”

34. Name the class “PlacesViewController” and make sure it inherits from NSObject

35. Locate the “views” folder, create a new folder in there called “places”, and save the file in that folder

36. Go to PlacesViewController.h and change the inheritance from NSObject to UIViewController

Now, what we need to do is copy some code over from ProfileViewController. This code will initialize the view controller, add it to the tab bar controller at the bottom of the app, and set the icons and subtitle appropriately.

37. Go to ProfileViewController.m and copy the init, viewDidLoad, and logoutButtonWasPressed functions.

38. Modify the init method’s first line to the following:

```
// call the superclass's initializer
self = [super init];
```

39. Change “Profile” to “Places” in the tab bar declaration:

```
// set the tab bar title information
UITabBarItem* tbi = [self tabBarItem];
[tbi setTitle:@"Places"];
UIImage* tb_image = [UIImage imageNamed:@"Places.png"];
[tbi setImage:tb_image];
```

40. Next, we need to modify the AppDelegate to display the new view controller that we’ve just created. Fortunately, we’ve isolated all of this in the _createLoggedInUI method, which we can modify as follows:

```
  // construct the profile view
  self.profileViewController = [[ProfileViewController alloc] init];
  UINavigationController *profileNavController =
[[UINavigationController alloc]
initWithRootViewController:self.profileViewController];

  // construct the places view
    self.placesViewController = [[PlacesViewController alloc] init];
    UINavigationController *placesNavController =
[[UINavigationController alloc]
initWithRootViewController:self.placesViewController];
```

```
  // add the tab view controllers to the tab bar controller, the last
item should be nil
  NSArray* controllers = [NSArray
arrayWithObjects:placesNavController, profileNavController, nil];
  self.tabBarController.viewControllers = controllers;
```

Finally, we need to create a new user interface for our view controller. Once again, we'll use the Interface Builder to do this.

41. Right-click on "places" and select "New File..."

42. Select "User Interface" (under iOS) in the left nav, and select "Empty" as the template type. The Device Family should be iPhone, name the file PlacesViewController and save it in the views/places folder where the PlacesViewController interface and implementation files are located.

43. Select the XIB file that has just been created

44. Drag and drop a UIView object onto the Interface Builder design surface

45. Drag and drop a label ("Search for Places:"), a text field, and a button ("Go") onto the view object

46. Drag and drop another label and position below the other elements

Once we've created the user interface, we need to wire it up to the implementation file.

47. Click on the "File's Owner" (orange) cube, then select PlacesViewController in the Class Owner field of the Identity Inspector

48. CTRL-click-drag from the "File's Owner" (orange) cube to the PlacesViewController and select "view"

49. Option-Click on PlacesViewController.h so that it is side-by-side with the XIB file

50. CTRL-click-drag from the text field to the empty line between @interface and @end in PlacesViewController.h

51. Create an Outlet, name it "placesSearchField"

52. CTRL-click-drag from the button to the empty line and create an Action and call it "searchButtonPressed"

53. CTRL-click-drag from the label (the one on the bottom) to the empty line and create an Outlet and call it "locationNameLabel"

54. Run the app and behold the mighty new tab!

Part 6: Incorporate the Core Location framework

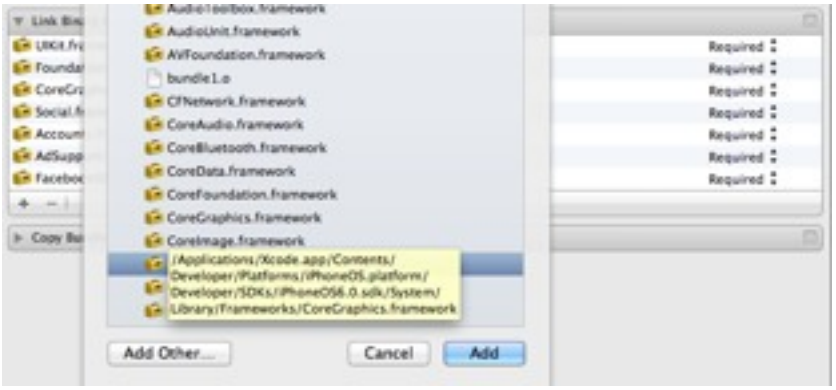
Now let's modify our app to grab the current location and display it on a map.

55. The first thing we need to do is include Core Location, the location library in iOS. Go to the Project Explorer and click on the blueprint looking icon at the very top.

56. Click on the target app (looks like the classic "App" icon) with the name "Scrumptious" next to it.

57. Select the "Build Phases" tab and open up the "LInk Binary with Libraries" section.

58. Click on the "+", find "CoreLocation.framework", and click "Add".



59. Then, we'll import it into the PlacesViewController.h header file:

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

60. Now, let's create a new property for the location manager. The location manager will do a couple of things for us. First, it will ask the user's permission for their location. Second, it will handle interfacing with the location hardware to get the latest coordinates.

```
@property (strong, nonatomic) CLLocationManager *locationManager;
```

61. Switch over to the PlacesViewController.m implementation file and synthesize the property

62. In the viewDidLoad method, we need to tell the location manager to start checking for location:

```
self.locationManager = [[CLLocationManager alloc] init];
self.locationManager.delegate = self;
```

```
    self.locationManager.desiredAccuracy =
kCLLocationAccuracyNearestTenMeters;
    // We don't want to be notified of small changes in location,
    // preferring to use our last cached results, if any.
    self.locationManager.distanceFilter = 50;
    [self.locationManager startUpdatingLocation];
```

63. We also need to do a little housekeeping and un-set the location manager delegate by implementing the dealloc function:

```
-(void)dealloc
{
    self.locationManager.delegate = nil;
}
```

64. Now we need to declare the core location delegate design pattern. First, we'll modify the class declaration in the PlacesViewController.h file:

```
@interface PlacesViewController : UIViewController
<CLLocationManagerDelegate>
```

65. Finally, we need to implement the core location delegate design pattern by implementing the didUpdateToLocation and didFailWithError functions:

```
// implement the CLLocationManagerDelegate
-(void)locationManager:(CLLocationManager *)manager
didUpdateLocations:(NSArray *)locations
{
    CLLocation *newLocation = [locations lastObject];
    NSLog(@"Got location: %f, %f",
          newLocation.coordinate.latitude,
          newLocation.coordinate.longitude);
}

- (void)locationManager:(CLLocationManager *)manager
       didFailWithError:(NSError *)error
{
    NSLog(@"%@", error);
}
```

66. Run the app, show the log message with the location.

Part 7: Call the Facebook PlacesViewController

Now we're going to wire up the Facebook PlacesViewController so that when we click the "Go" button, we see the familiar Places view from the Facebook iOS app.

67. Import the Facebook SDK in the PlacesViewController.h file (you may also remove the import statement from the PlacesViewController.m file, just for cleanliness)

68. Add a property for the FBPlacePickerViewController in the header file and synthesize it in the implementation file:

```
@property (strong, nonatomic) FBPlacePickerViewController
*placePickerController;
```

69. Make sure that the view is released in the viewDidUnload method:

```
-(void)viewDidUnload
{
    // not necessary in iOS 6.0
    self.placePickerController = nil;
}
```

70. In the searchButtonPressed method, instantiate the placePickerController if it hans't been already:

```
    if(!self.placePickerController) {
        self.placePickerController = [[FBPlacePickerViewController
alloc] initWithNibName:nil bundle:nil];
        self.placePickerController.title = @"Pick a Place";
    }
```

71. Now setup the location manager:

```
    self.placePickerController.locationCoordinate =
self.locationManager.location.coordinate;
    self.placePickerController.radiusInMeters = 1000;
    self.placePickerController.resultsLimit = 50;
```

72. And the search parameter comes from the search text field:

```
    self.placePickerController.searchText =
self.placesSearchField.text;
```

73. Finally, load the data and push the controller:

```
- (IBAction)searchButtonPressed:(id)sender
{
    if(!self.placePickerController) {
```

```
        self.placePickerController = [[FBPlacePickerViewController
alloc] initWithNibName:nil bundle:nil];
        self.placePickerController.title = @"Pick a Place";
    }

    self.placePickerController.locationCoordinate =
self.locationManager.location.coordinate;
    self.placePickerController.radiusInMeters = 1000;
    self.placePickerController.resultsLimit = 50;
    self.placePickerController.searchText =
self.placesSearchField.text;

    [self.placePickerController loadData];
    [self.navigationController
pushViewController:self.placePickerController animated:YES];
}
```

74. Run the application and take a look at the Facebook Place Picker Controller.

75. Okay, so how do we get the selected place back into the application? We use a delegate, of course!

76. Add the FBPlacePickerDelegate to the list of protocols supported by PlacesViewController in PlacesViewController.h:

```
@interface PlacesViewController : UIViewController
<CLLocationManagerDelegate, FBPlacePickerDelegate>
```

77. In the searchButtonPressed method, set the delegate of the PlacePickerViewController to be the current view controller:

```
    if(!self.placePickerController) {
        self.placePickerController = [[FBPlacePickerViewController
alloc] initWithNibName:nil bundle:nil];
        self.placePickerController.title = @"Pick a Place";
        self.placePickerController.delegate = self;
    }
```

78. Don't forget to nil out the delegate in the dealloc method:

```
    self.placePickerController.delegate = nil;
```

79. Finally, implement the delegate protocol in the PlacesViewController.m file:

```
-(void)placePickerViewControllerSelectionDidChange:
(FBPlacePickerViewController *)placePicker
```

```
{
    NSObject<FBGraphPlace> *place = placePicker.selection;

    self.locationNameLabel.text = place.name;
}
```

80. Run the application and savor the moment!

Part 8: Other Things to Try

You've come this far, so obviously you want to learn more!

Here are some things you may want to try (and for which sample code is available on Github):