

iPhone App Dev

Lesson 7

Source Codes

<https://github.com/makzan/ios-dev-course-example>

Contact

makzan@42games.net

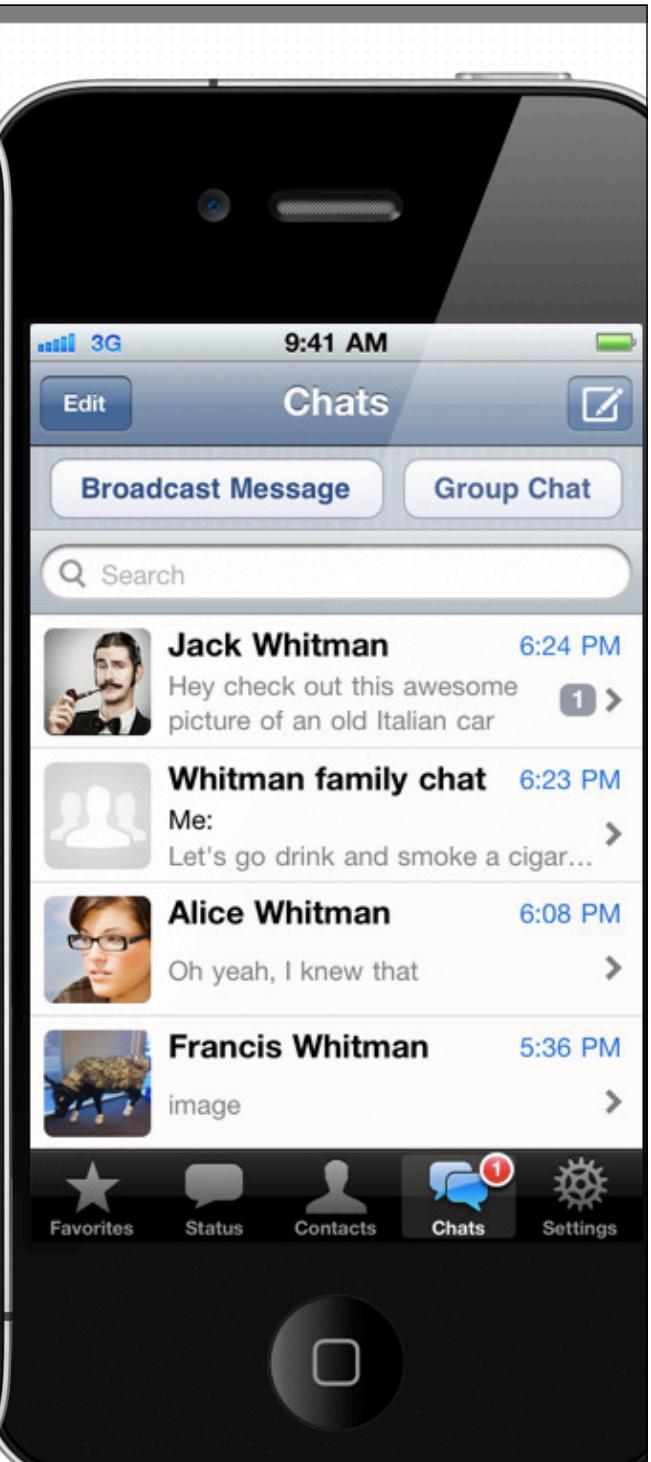
Reminder

- You need to develop an iOS app at the end of the course.

Summary

- Usage of table view
- Setting up table view
- Sections and rows
- Selecting table cell
- Choosing the table cell style
- Customizing the table cell accessory
- Using UISwitch in accessory view
- Displaying image in table view

Usage of table view

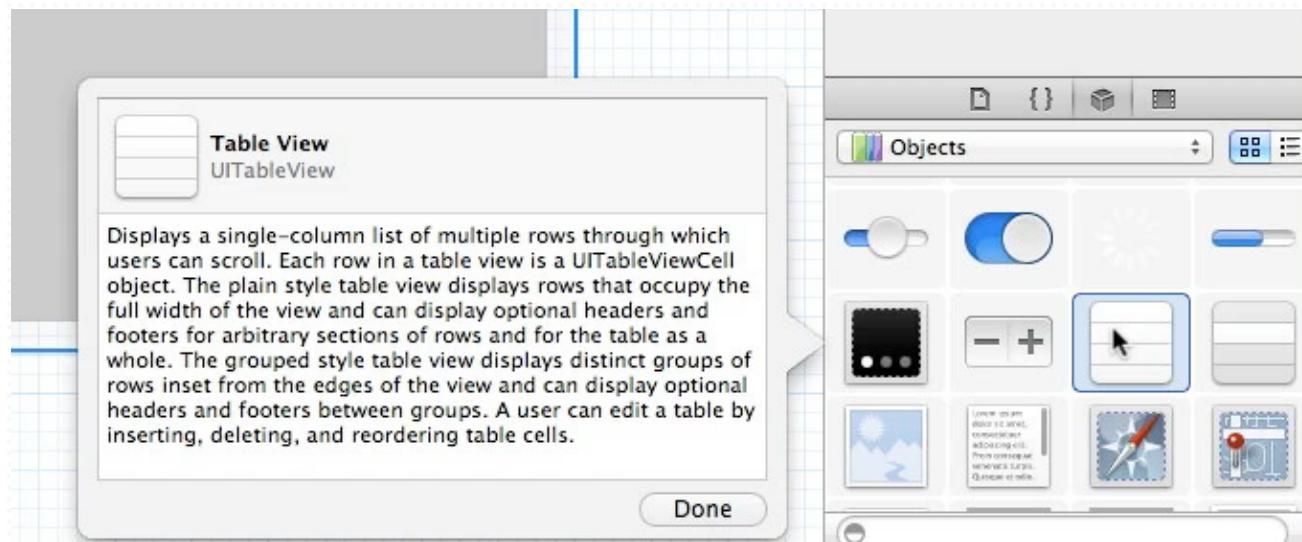


We find table view in almost every app.



Setting up table view

Setting up table view



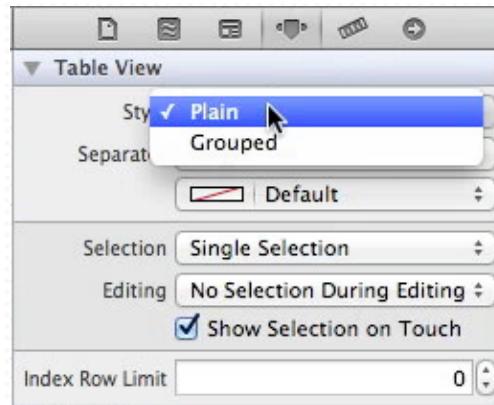
First, let's identify the table view in interface builder.

Setting up table view



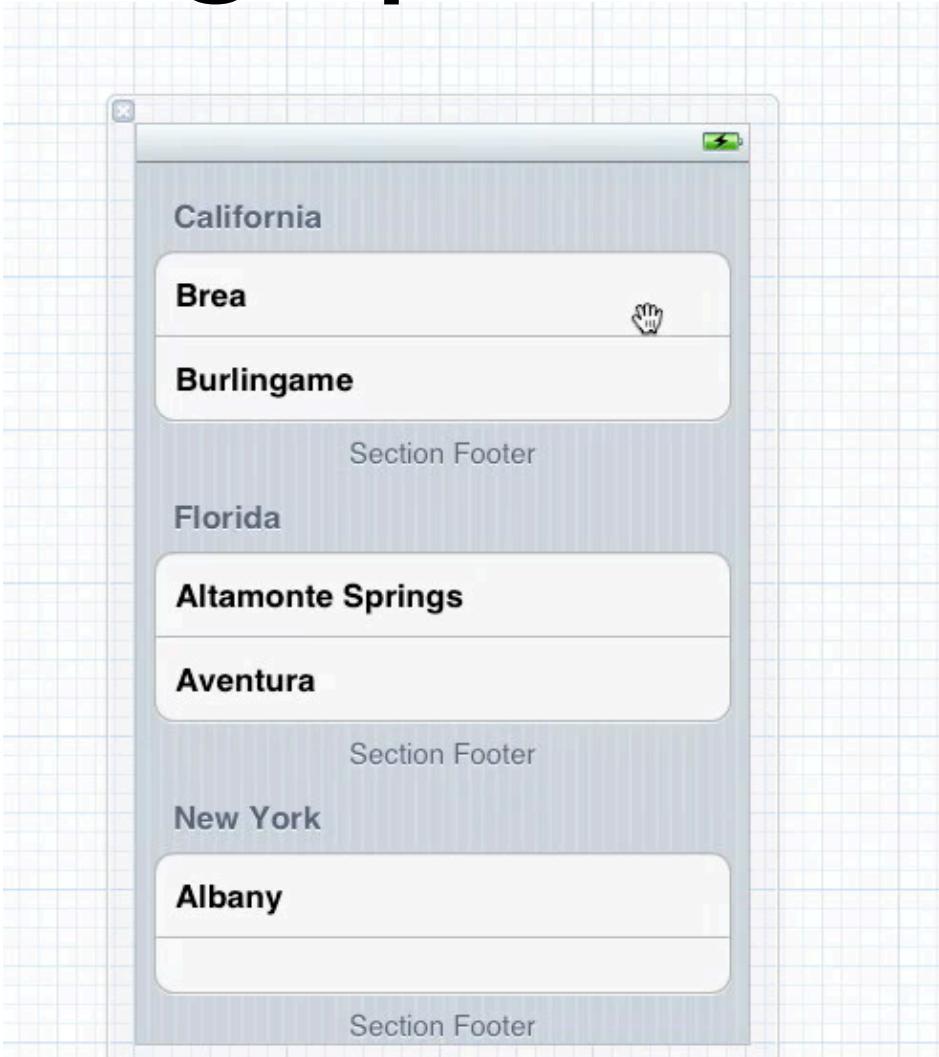
Drag the table view into the view.

Setting up table view



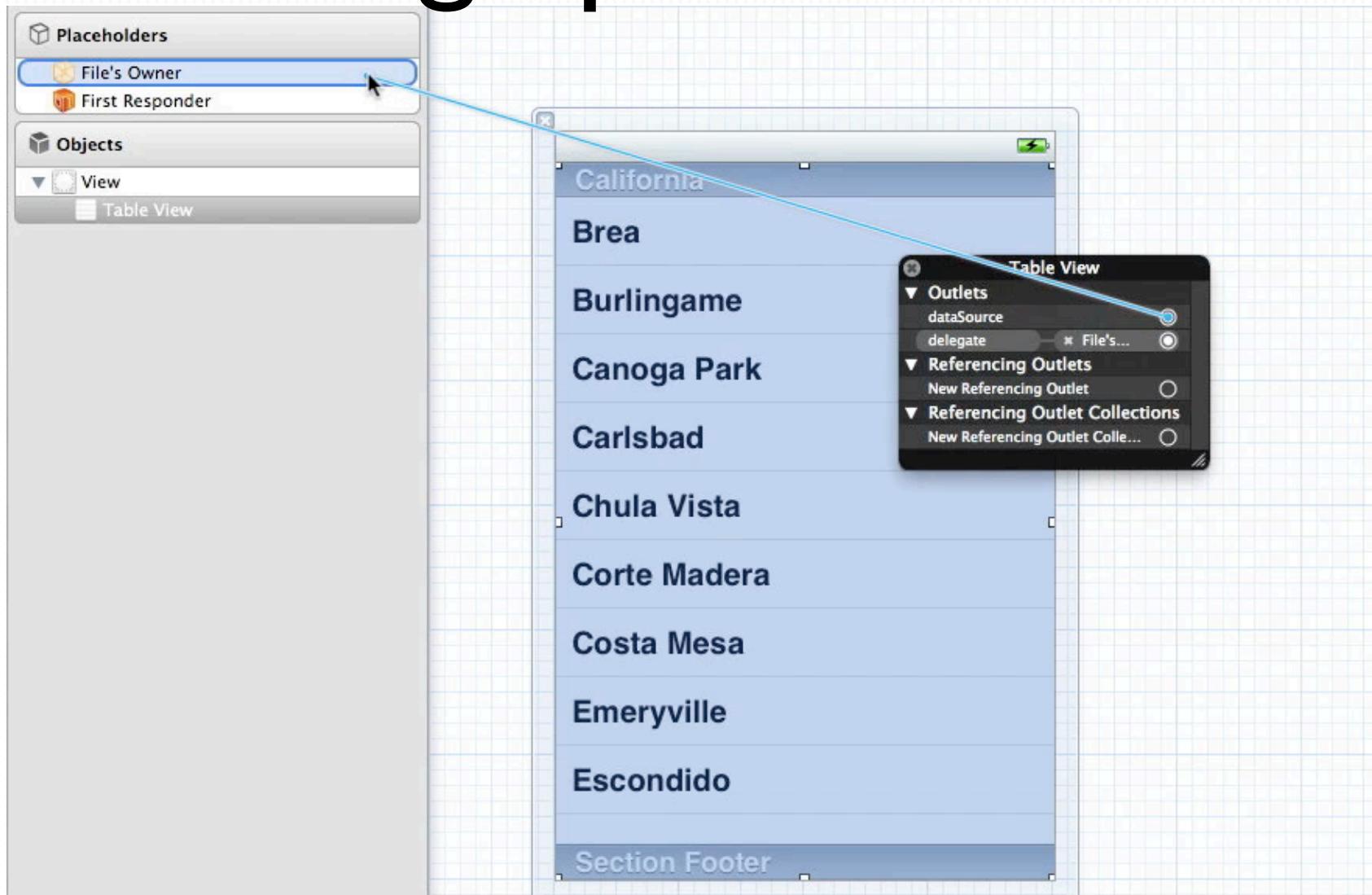
There are two styles in table view, Plain or Grouped.

Setting up table view



The default is plain and the above one is grouped style.

Setting up table view



Connect the delegate and dataSource to File's Owner

Setting up table view

ViewController.h

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

@interface ViewController : UIViewController <UITableViewDelegate, UITableViewDataSource>
@end
```

Include the table delegate and data source in header.

Setting up table view

ViewController.m

```
#pragma mark - Table Data Source  
  
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section  
{  
    return 3;  
}
```

Implement the method to set numbers of rows.

Setting up table view

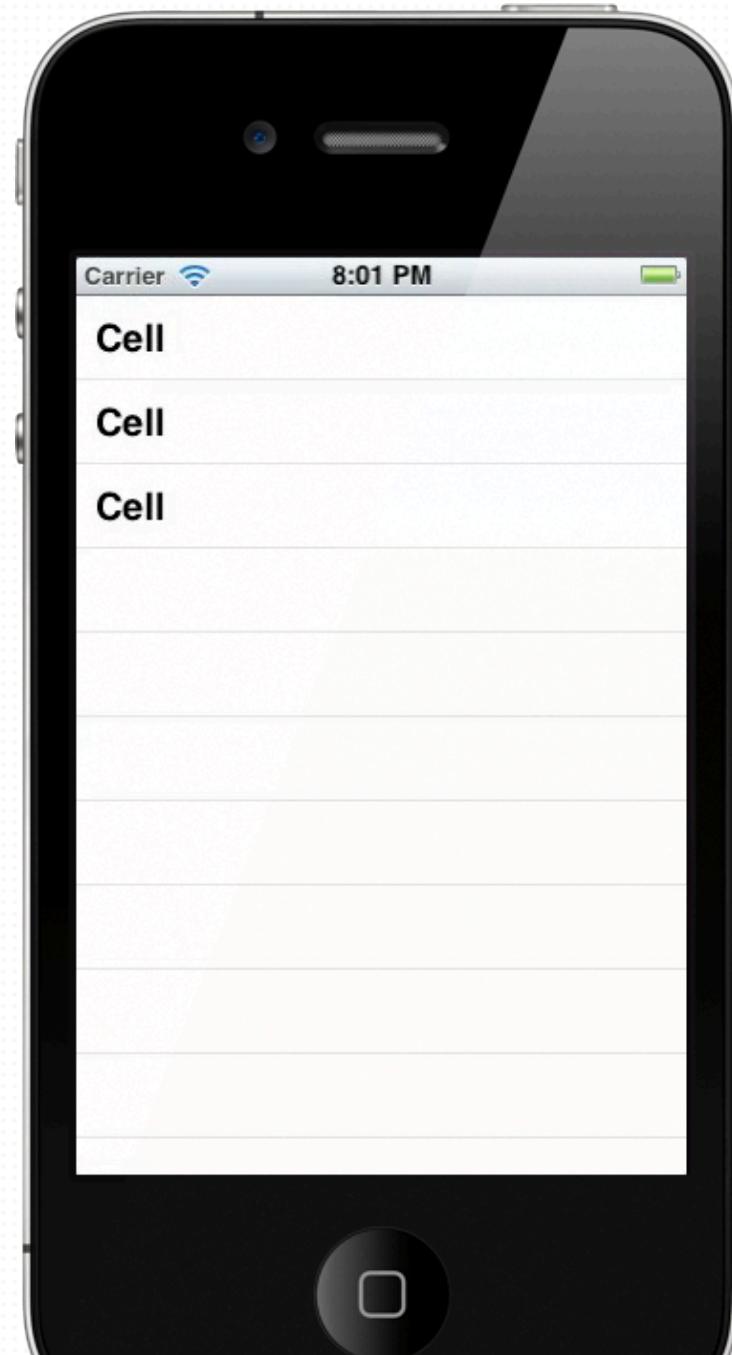
ViewController.m

```
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:@"NormalCell"];
    if (cell == nil)
    {
        cell = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault reuseIdentifier:@"NormalCell"];
    }

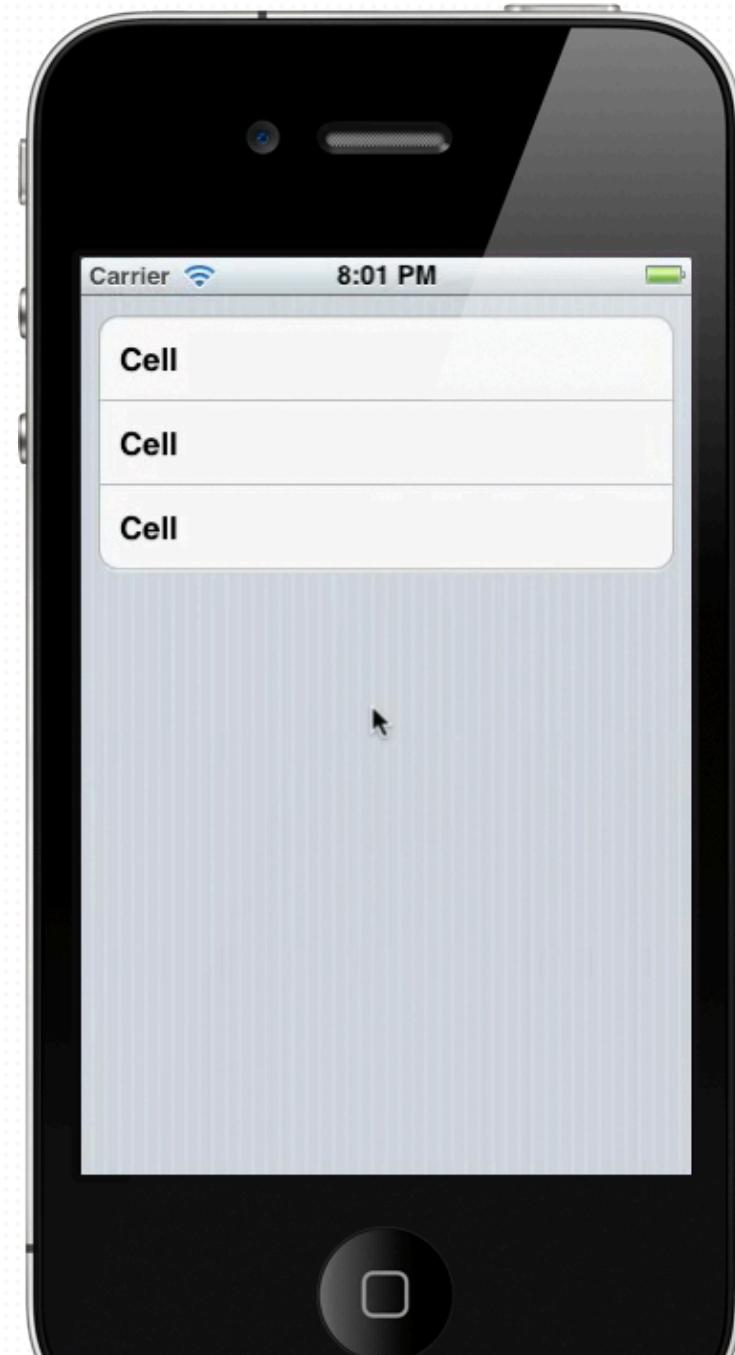
    cell.textLabel.text = @"Cell";
    return cell;
}
```

Implement the Table Cell in the above data source method.

The result in plain style.



The result in grouped style.



Section and Row

Sections and Rows

ViewController.m

```
- (NSString *)tableView:(UITableView *)tableView titleForHeaderInSection:  
    (NSInteger)section  
{  
    return @"Header Text";  
}
```

The section appears when we set the above data source method.

The result with section header



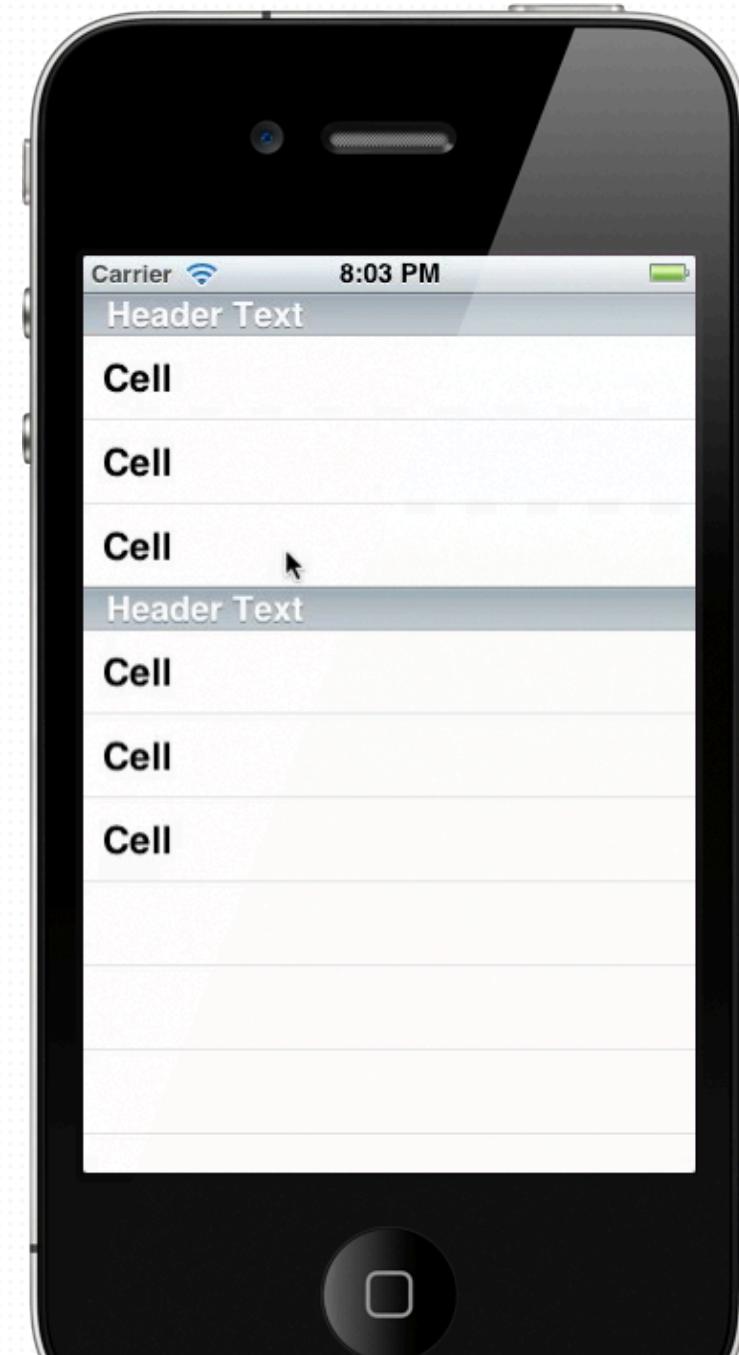
Sections and Rows

ViewController.m

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
    return 2;
}
```

We can also set how many sections in the table view

The result with 2 sections



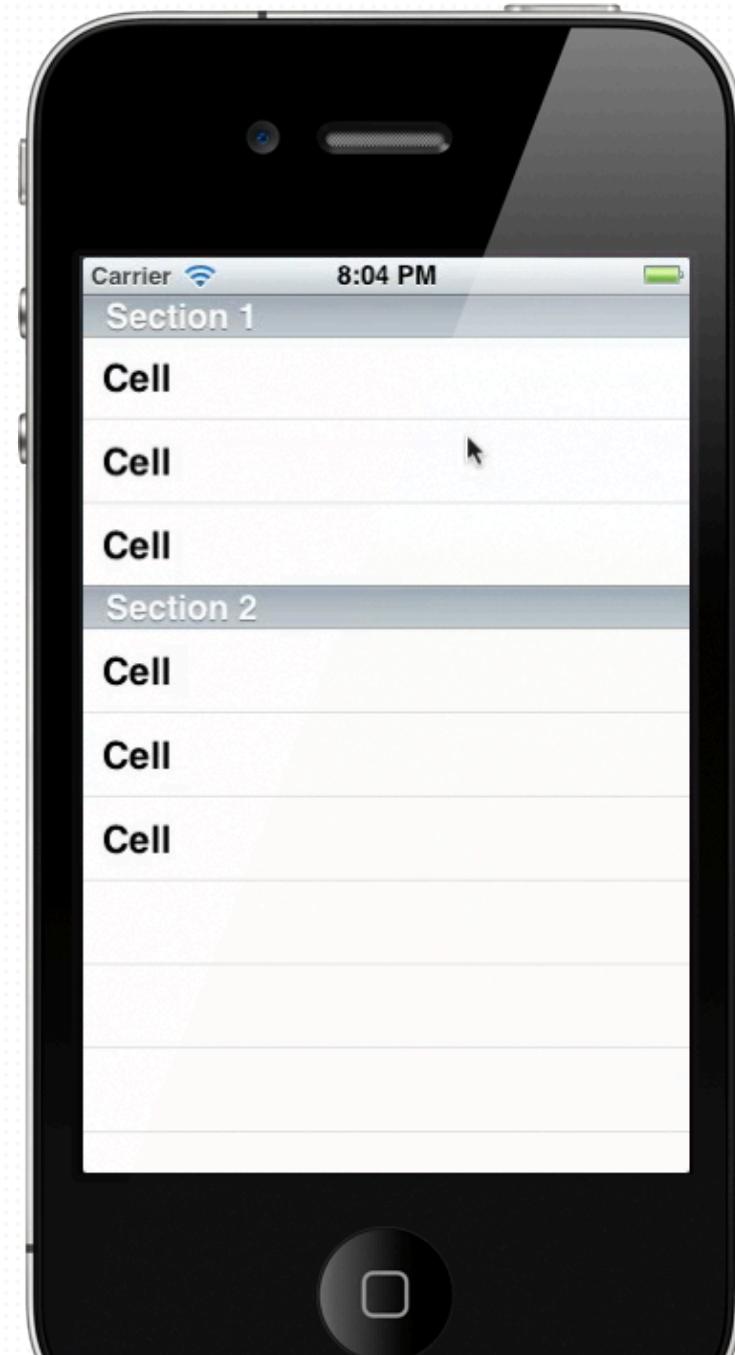
Sections and Rows

ViewController.m

```
- (NSString *)tableView:(UITableView *)tableView titleForHeaderInSection:(NSInteger)section
{
    if (section == 0)
    {
        return @"Section 1";
    }
    else if (section == 1)
    {
        return @"Section 2";
    }
    return @"";
}
```

We can set each section header

The result with 2
section headers



Sections and Rows

ViewController.m

```
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:@"NormalCell"];
    if (cell == nil)
    {
        cell = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault reuseIdentifier:@"NormalCell"];
    }

    if (indexPath.row == 0)
    {
        cell.textLabel.text = @"Cell 1";
    }
    else if (indexPath.row == 1)
    {
        cell.textLabel.text = @"Cell 2";
    }
    else if (indexPath.row == 2)
    {
        cell.textLabel.text = @"Cell 3";
    }

    return cell;
}
```

The cell implementation method becomes more complicated to display different cell content.

The result with different cell content.



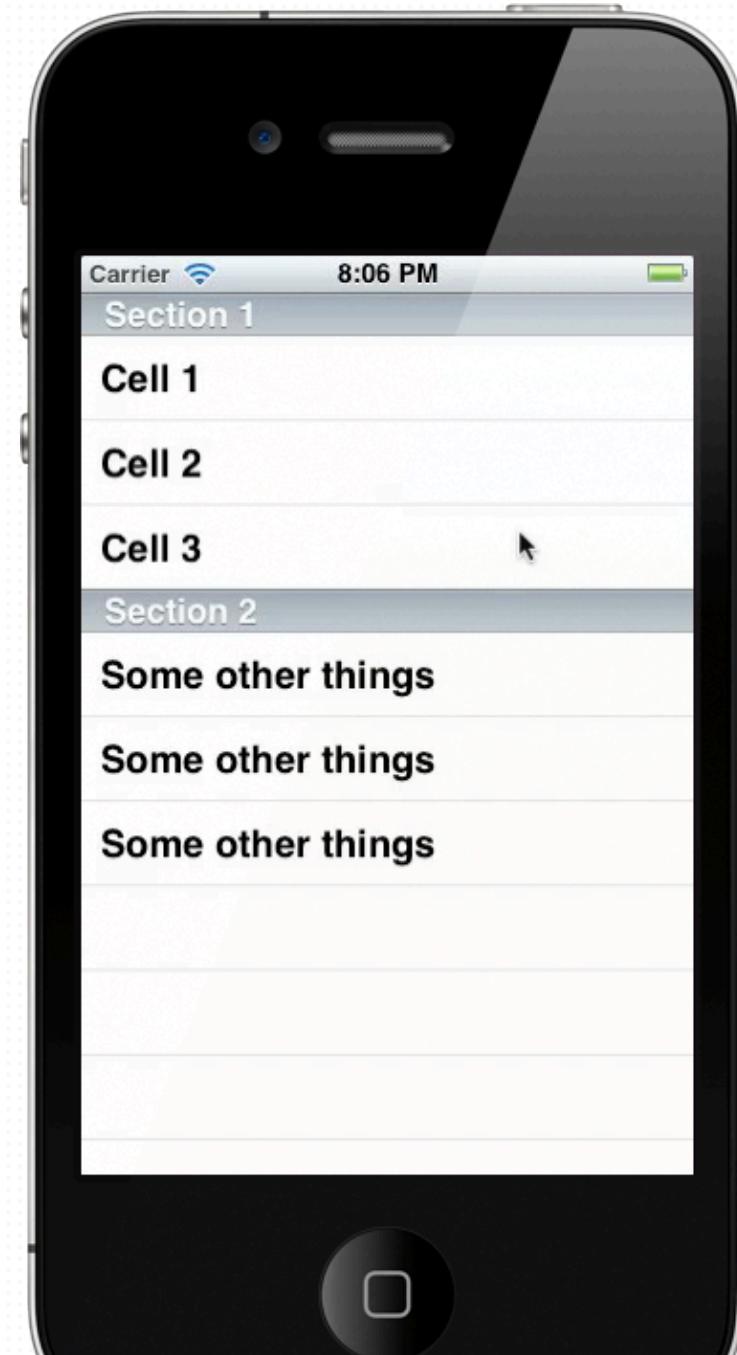
Sections and Rows

ViewController.m

```
if (indexPath.section == 0)
{
    if (indexPath.row == 0)
    {
        cell.textLabel.text = @"Cell 1";
    }
    else if (indexPath.row == 1)
    {
        cell.textLabel.text = @"Cell 2";
    }
    else if (indexPath.row == 2)
    {
        cell.textLabel.text = @"Cell 3";
    }
}
else if (indexPath.section == 1)
{
    cell.textLabel.text = @"Some other things";
}
```

We may also set content based on the section index

The result with different cell content.



Sections and Rows

ViewController.m

```
@interface ViewController ()  
@property (nonatomic, retain) NSMutableArray *dataArray;  
@end  
  
@implementation ViewController  
@synthesize dataArray = _dataArray;  
  
- (void)viewDidLoad  
{  
    [super viewDidLoad];  
    // Do any additional setup after loading the view, typically from a nib.  
  
    self.dataArray = [[NSMutableArray alloc] initWithCapacity:10];  
    [self.dataArray addObject:@"Record 1"];  
    [self.dataArray addObject:@"Record 2"];  
    [self.dataArray addObject:@"Record 3"];  
}
```

Let's try using some dynamic content. Create an Array with some string content.

Sections and Rows

ViewController.m

```
        ,  
    else if (indexPath.section == 1)  
{  
    cell.textLabel.text = [selfdataArray objectAtIndex:indexPath.row];  
}
```

Replace the table cell text to the array content.

The result with cell content
from array.



Sections and Rows

ViewController.m

```
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
{
    if (section == 0)
    {
        return 3;
    }
    else if (section == 1)
    {
        return [self.dataArray count];
    }

    return 0;
}
```

We can set the number of rows according to the object count in the data array

The result with many records in the data array.



The same code base with grouped style



Selecting table cell

Selecting table cell

ViewController.m

```
#pragma mark - Table View Delegates  
- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:  
    (NSIndexPath *)indexPath  
{  
    [tableView deselectRowAtIndexPath:indexPath animated:YES];  
}
```

We can know the table cell selection with the above delegate method.

The cell with selection and deselection effect



Table cell style

Table cell style

ViewController.m

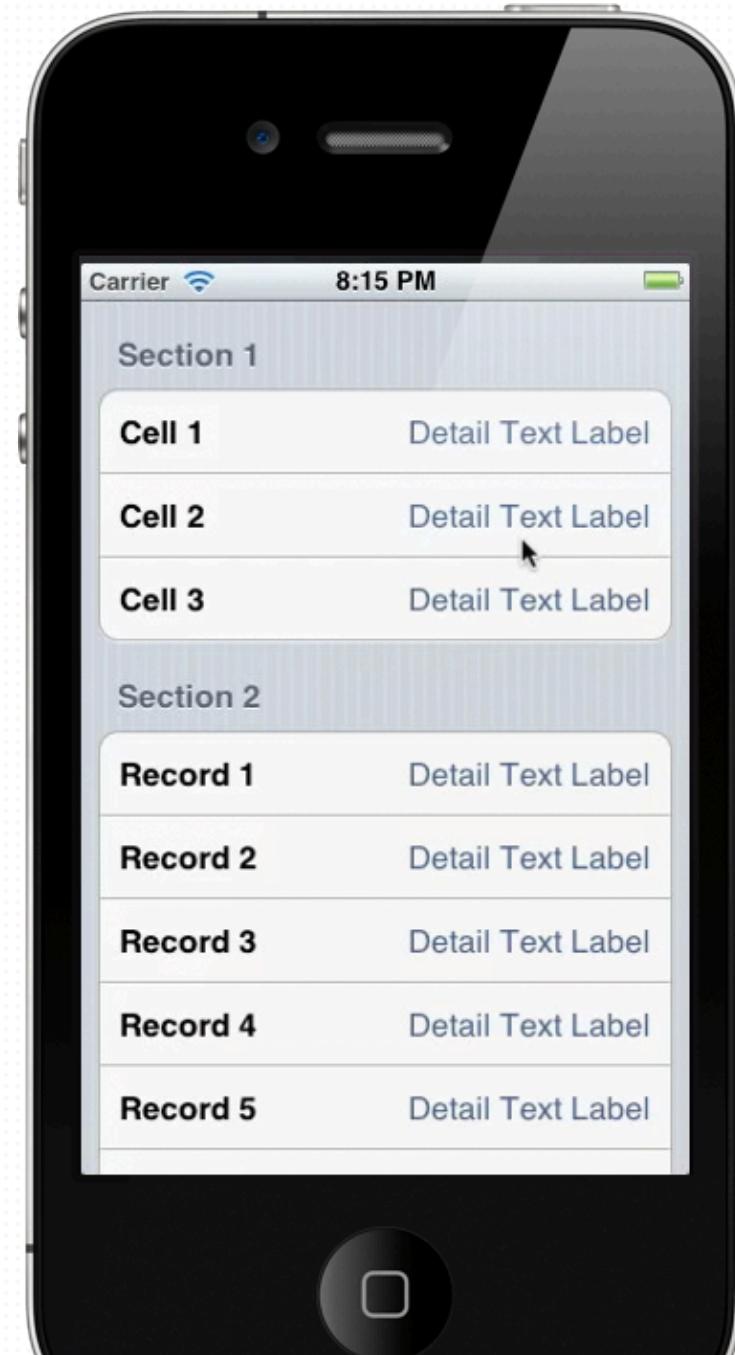
```
{  
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:@"NormalCell"];  
    if (cell == nil)  
    {  
        cell = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleSubtitle reuseIdentifier:  
            @"NormalCell"];  
    }  
  
    if (indexPath.section == 0)  
    {  
        if (indexPath.row == 0)  
        {  
            cell.textLabel.text = @"Cell 1";  
        }  
  
        cell.detailTextLabel.text = @"Detail Text Label";  
    }  
}
```

There are 4 cells styles. We also set a detail text label to preview the cell styles.

The result with subtitle cell style



The result with value | cell style



The result with value 2
cell style



Table cell accessory

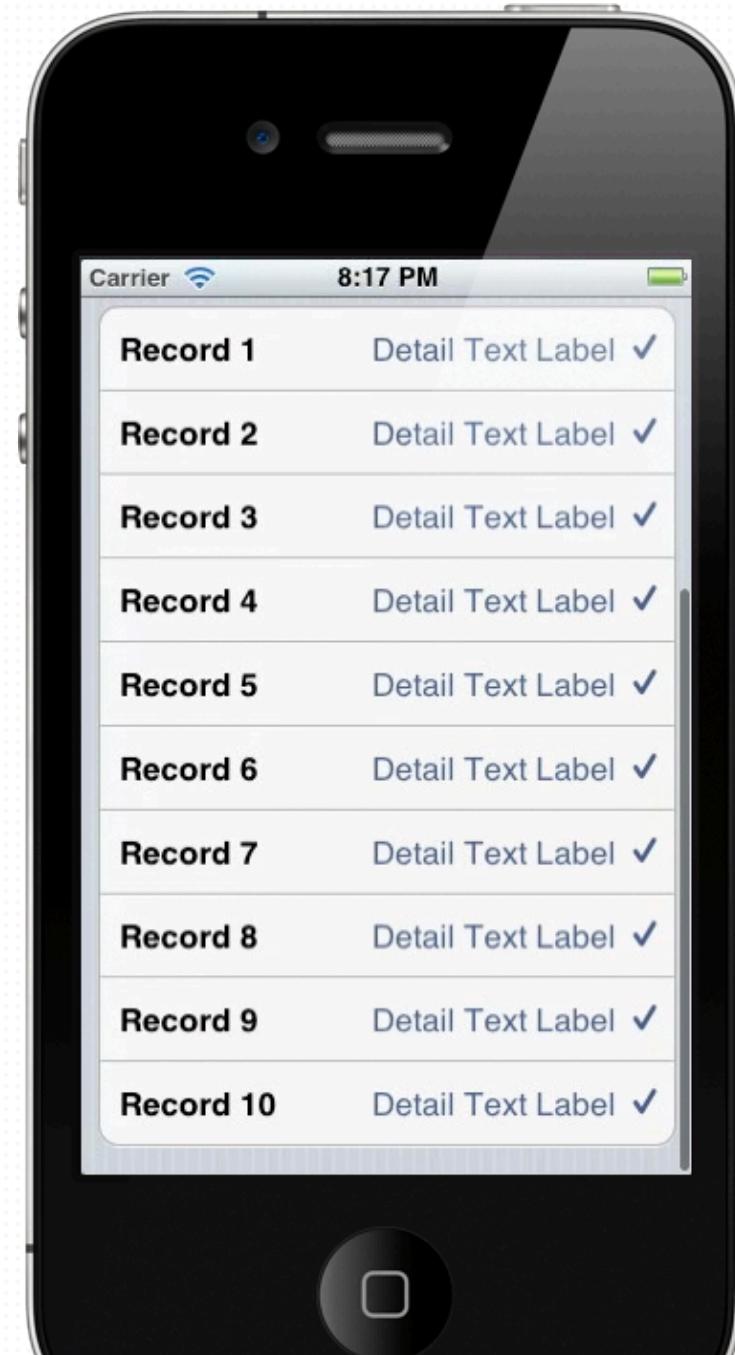
Table cell accessory

ViewController.m

```
cell.accessoryType = UITableViewCellAccessoryCheckmark
[K UITableViewCellAccessoryType UITableViewCellAccessoryCheckmark
[K UITableViewCellAccessoryType UITableViewCellAccessoryDetailDisclosureButton
[K UITableViewCellAccessoryType UITableViewCellAccessoryDisclosureIndicator
[K UITableViewCellAccessoryType UITableViewCellAccessoryNone (NSI
    if (section == 0)
```

We can set the accessoryType with several default option

The result with check mark accessory



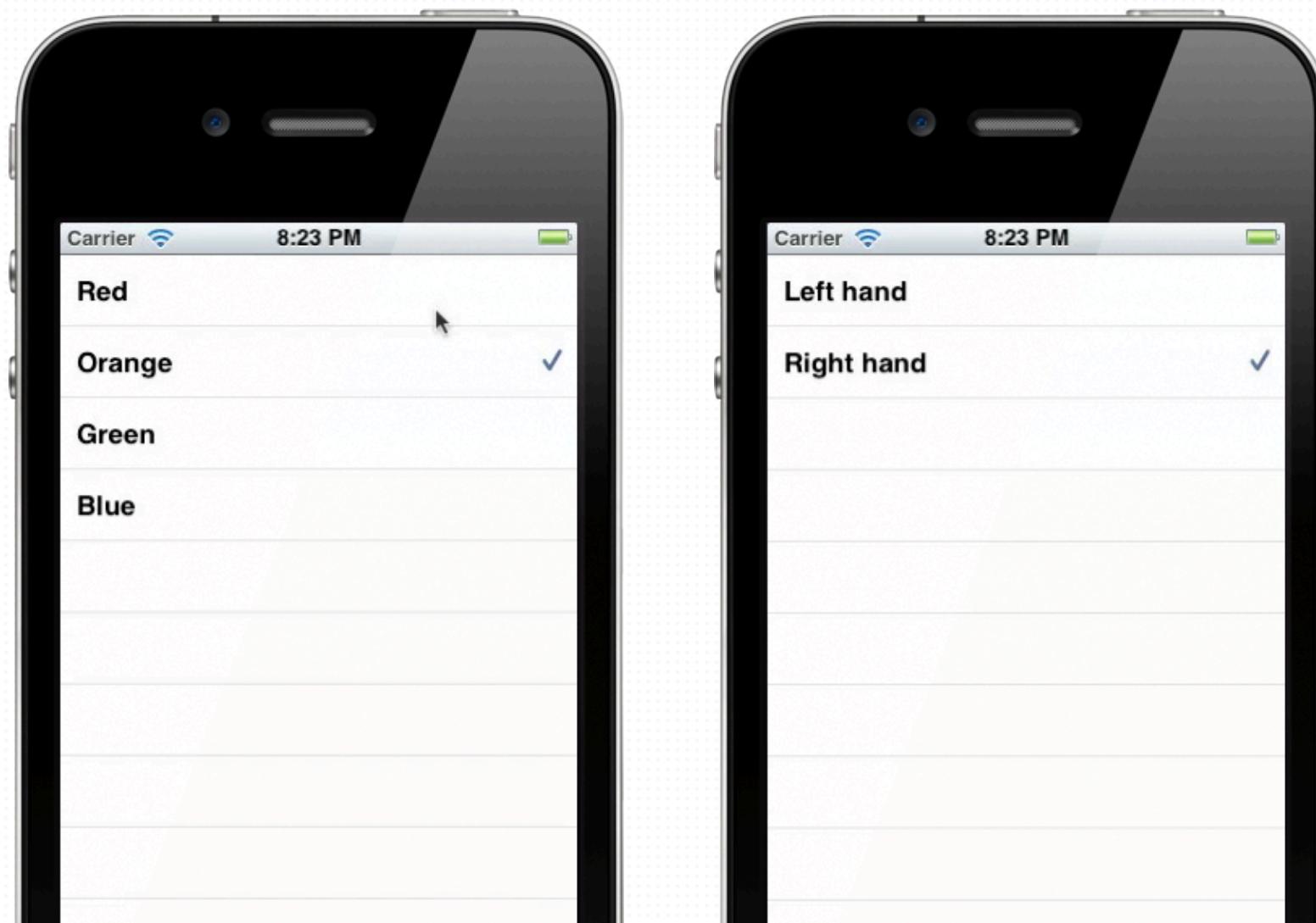
The result with detail disclosure button accessory



The result with disclosure indicator accessory



Example: Choosing Options in Table View



Option Choosing

ViewController.m

```
@implementation ViewController {  
    int chosenIndex;  
}
```

Create an instance variable to store the chosen index

Option Choosing

ViewController.m

```
#pragma mark - Table Data Source
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
{
    return 2;
}
```

Set the number of rows to 2 because we are showing two options now.

Option Choosing

ViewController.m

```
// reset the check mark first
cell.accessoryType = UITableViewCellAccessoryNone;

// and show the mark if this cell is the one we want.
if (chosenIndex == indexPath.row)
{
    cell.accessoryType = UITableViewCellAccessoryCheckmark;
}

if (indexPath.row == 0)
{
    cell.textLabel.text = @"Left hand";
}
else if (indexPath.row == 1)
{
    cell.textLabel.text = @"Right hand";
}
```

We change the cell implementation method to the above

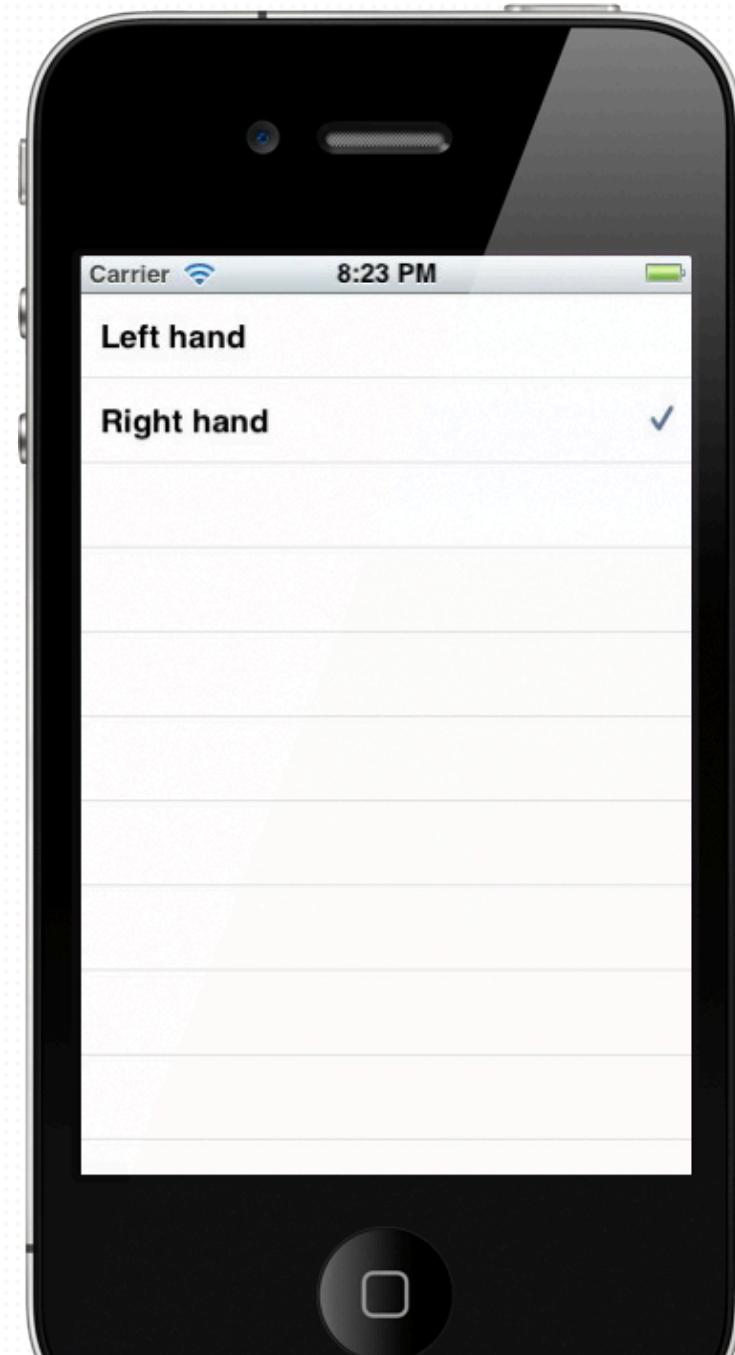
Option Choosing

ViewController.m

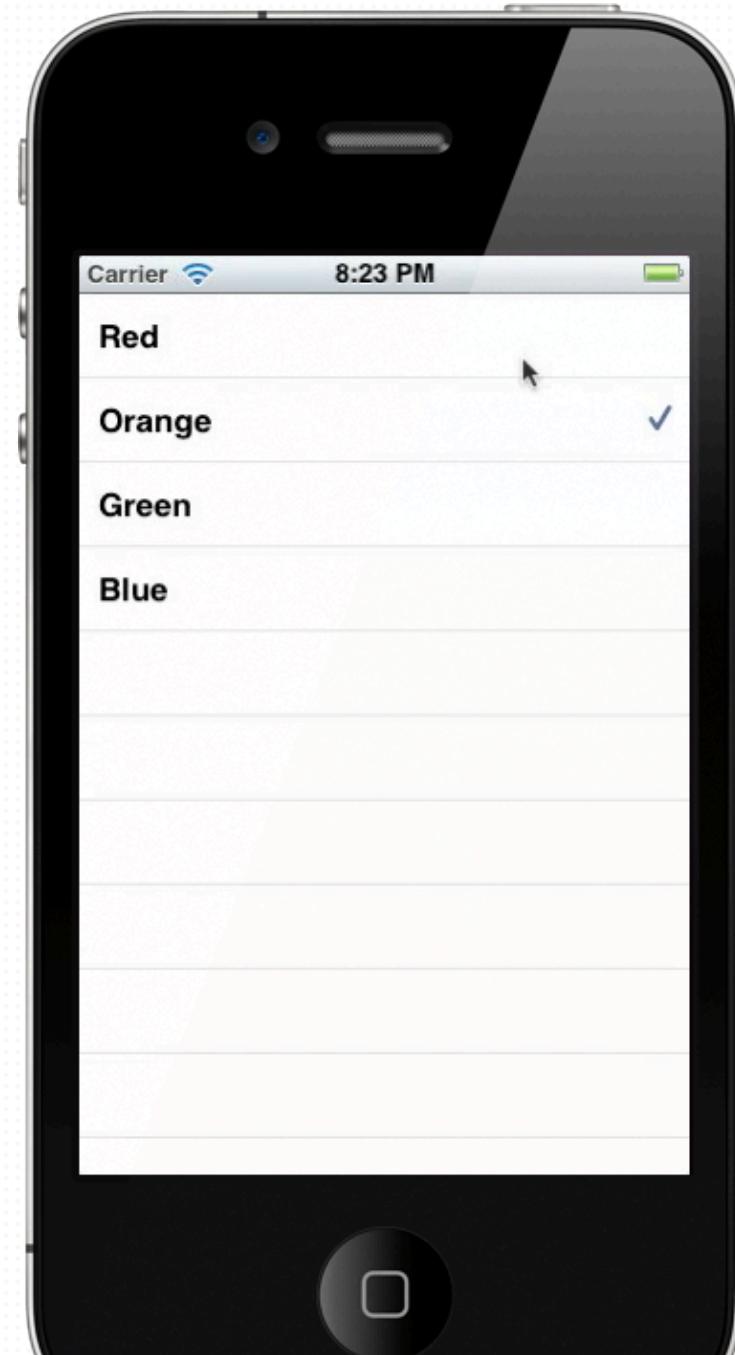
```
#pragma mark - Table View Delegates  
  
- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath  
{  
    [tableView deselectRowAtIndexPath:indexPath animated:YES];  
  
    chosenIndex = indexPath.row;  
  
    [tableView reloadData];  
}
```

When selecting the cell, we store the index and refresh the table view

The result with two options



We can extend the example
with more options.



UISwitch view inside cell

UISwitch inside cell

ViewController.m

```
else if (indexPath.row == 2)
{
    cell.textLabel.text = @"Cell 3";
}

else if (indexPath.section == 1)
{
    cell.textLabel.text = [selfdataArray objectAtIndex:indexPath.row];

    UISwitch *switchView = [[UISwitch alloc] init];
    cell.accessoryView = switchView;

    [switchView release];
}

return cell;
}
```

Instead of the default accessory options, we can directly put a view in the accessory view

The cell with switch view



UISwitch inside cell

ViewController.m

```
        }

    else if (indexPath.section == 1)
    {
        cell.textLabel.text = [selfdataArray objectAtIndex:indexPath.row];

        UISwitch *switchView = [[UISwitch alloc] init];
        NSLog(@"retain count: %d", [switchView retainCount]);
        cell.accessoryView = switchView;
        NSLog(@"retain count: %d", [switchView retainCount]);

        [switchView release];
        NSLog(@"retain count: %d", [switchView retainCount]);
    }

    return cell;
}
```

Will the ‘release’ cause problem? Let’s check if the accessoryView retains the view

UISwitch inside cell

```
All Output ▾ Clear □□□  
2012-06-08 20:28:40.226 TableViewDemo[5639:f803] retain count: 1  
2012-06-08 20:28:40.228 TableViewDemo[5639:f803] retain count: 2  
2012-06-08 20:28:40.229 TableViewDemo[5639:f803] retain count: 1  
2012-06-08 20:28:40.230 TableViewDemo[5639:f803] retain count: 1  
2012-06-08 20:28:40.231 TableViewDemo[5639:f803] retain count: 2  
2012-06-08 20:28:40.231 TableViewDemo[5639:f803] retain count: 1  
2012-06-08 20:28:40.232 TableViewDemo[5639:f803] retain count: 1  
2012-06-08 20:28:40.233 TableViewDemo[5639:f803] retain count: 2  
2012-06-08 20:28:40.233 TableViewDemo[5639:f803] retain count: 1  
2012-06-08 20:28:40.234 TableViewDemo[5639:f803] retain count: 1  
2012-06-08 20:28:40.235 TableViewDemo[5639:f803] retain count: 2  
2012-06-08 20:28:40.235 TableViewDemo[5639:f803] retain count: 1  
2012-06-08 20:28:40.236 TableViewDemo[5639:f803] retain count: 1  
2012-06-08 20:28:40.237 TableViewDemo[5639:f803] retain count: 2  
2012-06-08 20:28:40.237 TableViewDemo[5639:f803] retain count: 1  
2012-06-08 20:28:40.238 TableViewDemo[5639:f803] retain count: 1  
2012-06-08 20:28:40.239 TableViewDemo[5639:f803] retain count: 2  
2012-06-08 20:28:40.275 TableViewDemo[5639:f803] retain count: 1
```

The retain count shows that accessoryView retains the view and increases the retain count to 2. So we are safe here.

UISwitch inside cell

ViewController.m

```
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    NSString *identifier = @"NormalCell";

    if (indexPath.section == 1)
    {
        identifier = @"SwitchCell";
    }

    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:identifier];
    if (cell == nil)
    {
        cell = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleValue1 reuseIdentifier:identifier];
    }
}
```

Since we have two types of table cell now, we need two identifier for the cell reuse

Setup IBAction in code

ViewController.m

```
    }
    else if (indexPath.section == 1)
    {
        cell.textLabel.text = [selfdataArray objectAtIndex:indexPath.row];

        UISwitch *switchView = [[UISwitch alloc] init];
        [switchView addTarget:self action:@selector(switchViewToggled:) forControlEvents:UIControlEventValueChanged];
        cell.accessoryView = switchView;

        [switchView release];
    }
}
```

We want to track the valueChanged IBAction. We do not have interface builder, so we setup the IBAction in code.

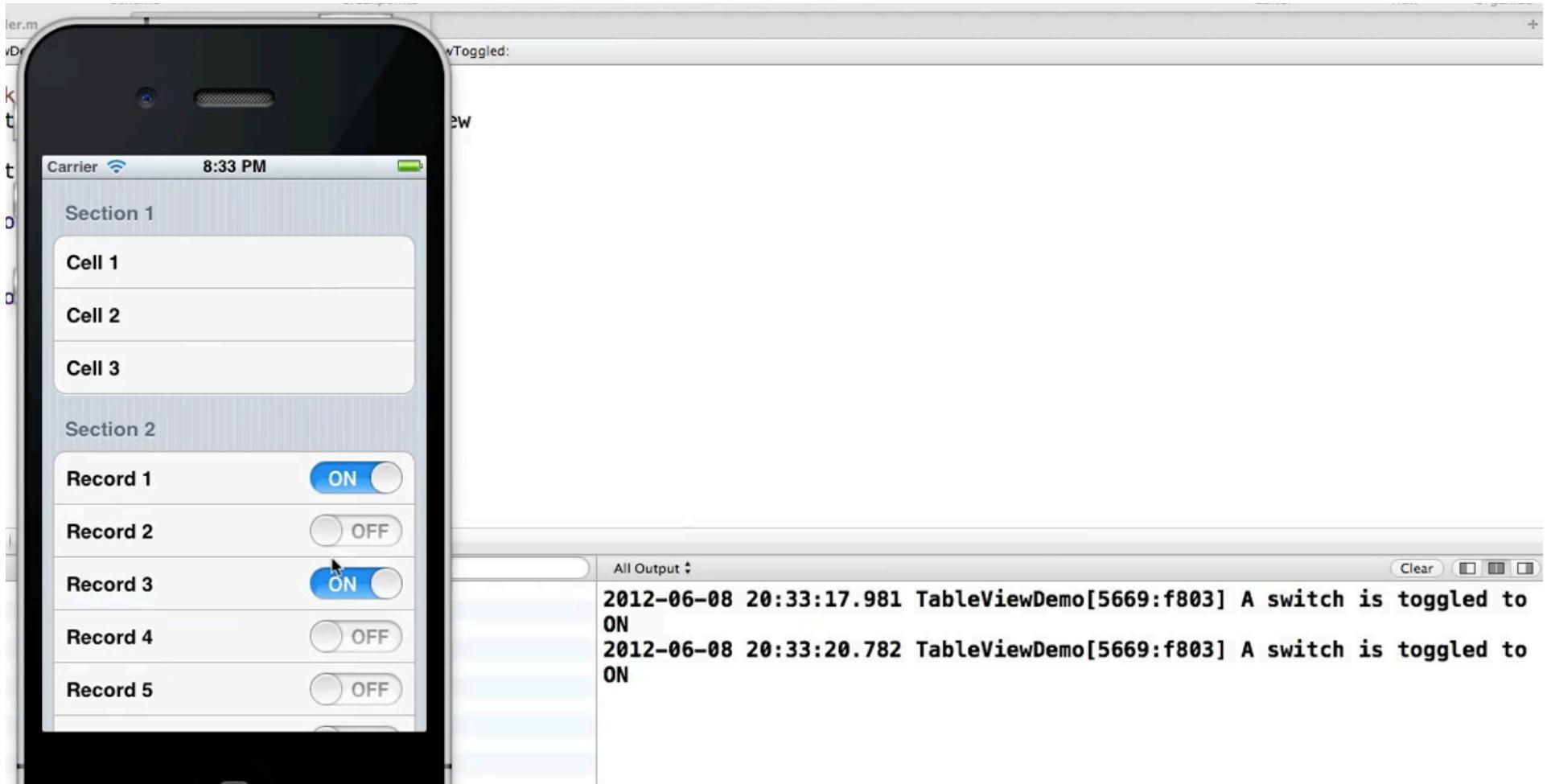
UISwitch inside cell

ViewController.m

```
#pragma mark - Switch View Actions
- (void)switchViewToggled:(UISwitch*)switchView
{
    if (switchView.on)
    {
        NSLog(@"A switch is toggled to ON");
    }
    else {
        NSLog(@"A switch is toggled to OFF");
    }
}
```

We setup the IBAction method to track the switch

UISwitch inside cell



Now we can know a switch is toggled. But we do not know which one.

UISwitch inside cell

ViewController.m

```
    }

}

else if (indexPath.section == 1)
{
    cell.textLabel.text = [selfdataArray objectAtIndex:indexPath.row];

    UISwitch *switchView = [[UISwitch alloc] init];
    [switchView addTarget:self action:@selector(switchViewToggled:) forControlEvents:UIControlEventValueChanged];
    switchView.tag = indexPath.row;
    cell.accessoryView = switchView;

    [switchView release];
}

return cell;
}
```

There is a ‘tag’ property in every UIView. We can assign any integer on it. Let’s assign the cell row index.

UISwitch inside cell

ViewController.m

```
#pragma mark - Switch View Actions
- (void)switchViewToggled:(UISwitch*)switchView
{
    if (switchView.on)
    {
        NSLog(@"switch %d is toggled to ON", switchView.tag);
    }
    else {
        NSLog(@"switch %d is toggled to OFF", switchView.tag);
    }
}
```

And revisit the switch view IBAction method with the view tag value

UISwitch inside cell

The screenshot shows a Xcode interface with a simulated iPhone screen on the left and a code editor and log viewer on the right.

Simulated Device Screen: Displays a UITableView with two sections. Section 1 contains three cells labeled "Cell 1", "Cell 2", and "Cell 3". Section 2 contains five cells labeled "Record 1" through "Record 5". Each cell contains a UISwitch. The switches for Record 1, Record 3, and Record 4 are currently set to "ON".

Code Editor (wController.m):

```
turn @

a mark
d)tabl
ableVi
a mark
d)swit
(swit
NSLo
Carrier 8:34 PM
Section 1
Cell 1
Cell 2
Cell 3
Section 2
Record 1 ON
Record 2 OFF
Record 3 ON
Record 4 OFF
Record 5 OFF
```

```
tableViewSelectRowAtIndexPath:(NSIndexPath *)indexPath
    withRowAnimation:YES];
new
switchView.tag);
```

Log Viewer: Shows the output of the application's NSLog statements.

```
All Output
2012-06-08 20:34:05.317 TableViewDemo[5705:f803] switch 0 is toggled to ON
2012-06-08 20:34:07.601 TableViewDemo[5705:f803] switch 2 is toggled to ON
2012-06-08 20:34:08.299 TableViewDemo[5705:f803] switch 2 is toggled to OFF
2012-06-08 20:34:08.979 TableViewDemo[5705:f803] switch 2 is toggled to ON
```

The result showing the switch index when toggling

UISwitch inside cell Correct Way

```
self.switchStatuses = [[NSMutableArray alloc] initWithCapacity:10];
for (int i=0;i<10;i++)
{
    [self.switchStatuses addObject:@(NO)];
}
```

In the viewDidLoad method, we init an array with 10 NOs.

UISwitch inside cell Correct Way

```
switchView.on = [[self.switchStatuses objectAtIndex:indexPath.row] boolValue];
```

When preparing the cell with switch view, we set the switch on property according to the bool value.

UISwitch inside cell Correct Way

```
#pragma mark - Switch View Actions
- (void)switchViewToggled:(UISwitch*)switchView
{
    if (switchView.on)
    {
        NSLog(@"switch %d is toggled to ON", switchView.tag);
    }
    else {
        NSLog(@"switch %d is toggled to OFF", switchView.tag);
    }

    [self.switchStatuses replaceObjectAtIndex:switchView.tag withObject:@(switchView.on)];
}
```

When the switch is toggled, we save the boolean
into the switch statuses array.

Displaying image in cell

Displaying image in cell

ViewController.m

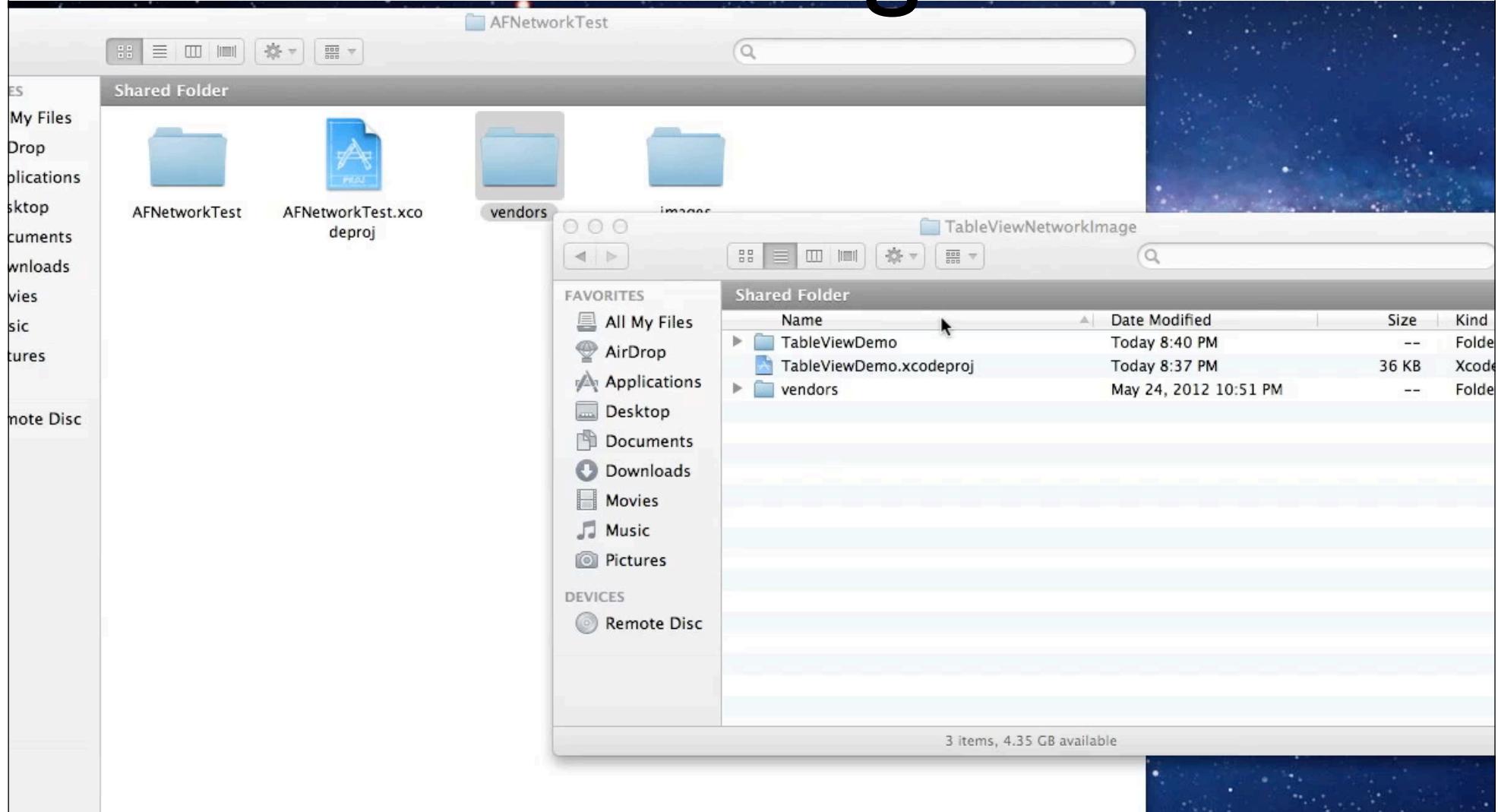
```
cell.imageView.image = [UIImage imageNamed:@"face.png"];
```

There is a default image property showing image on the left hand side of the cell

The result with local cell image

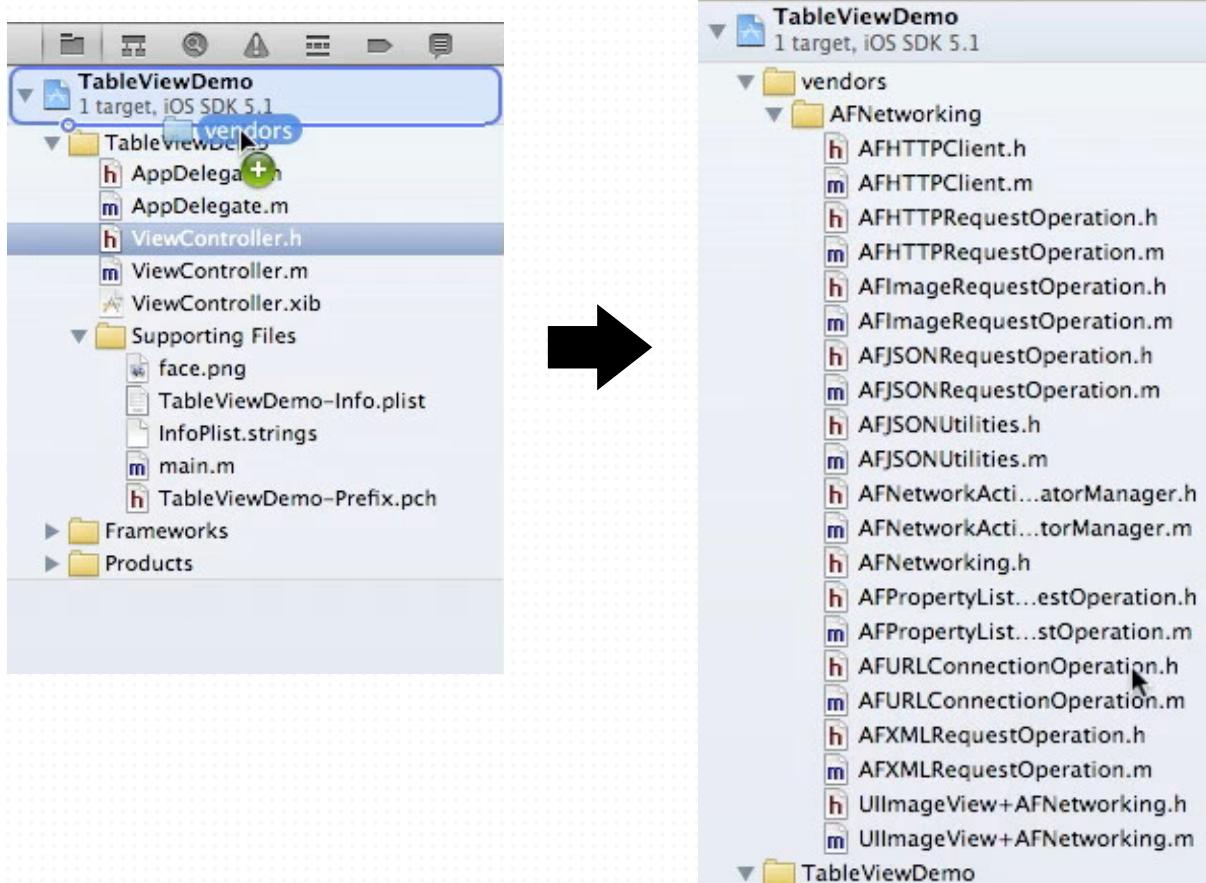


Network image in cell



To display networking image, we need the AFNetworking

Network image in cell



Add the AFNetworking files into XCode project

Network image in cell

ViewController.m

```
#import "ViewController.h"
#import "UIImageView+AFNetworking.h"

@interface ViewController : UIViewController
@property (nonatomic, retain) NSMutableArray *dataArray;
@end
```

Import the UIImageView+AFNetworking.h file

Network image in cell

ViewController.m

```
NSURL * imageURL = [NSURL URLWithString:@"http://placekitten.com/80/80.png"];
[cell.imageView setImageWithURL:imageURL];
```

Use the AFNetworking method to set the cell image view
with an image URL

The result with
networking image



The same code base with plain table style



Exercise

- Integrate table view in your app
 - ▶ App setting view
 - ▶ List of choices
 - ▶ Groups of app buttons
 - ▶ Displaying dynamic content in table cell
- What other table delegate or data source methods we can use?
- Table view is the most important element in iOS dev. Explore it and master it!