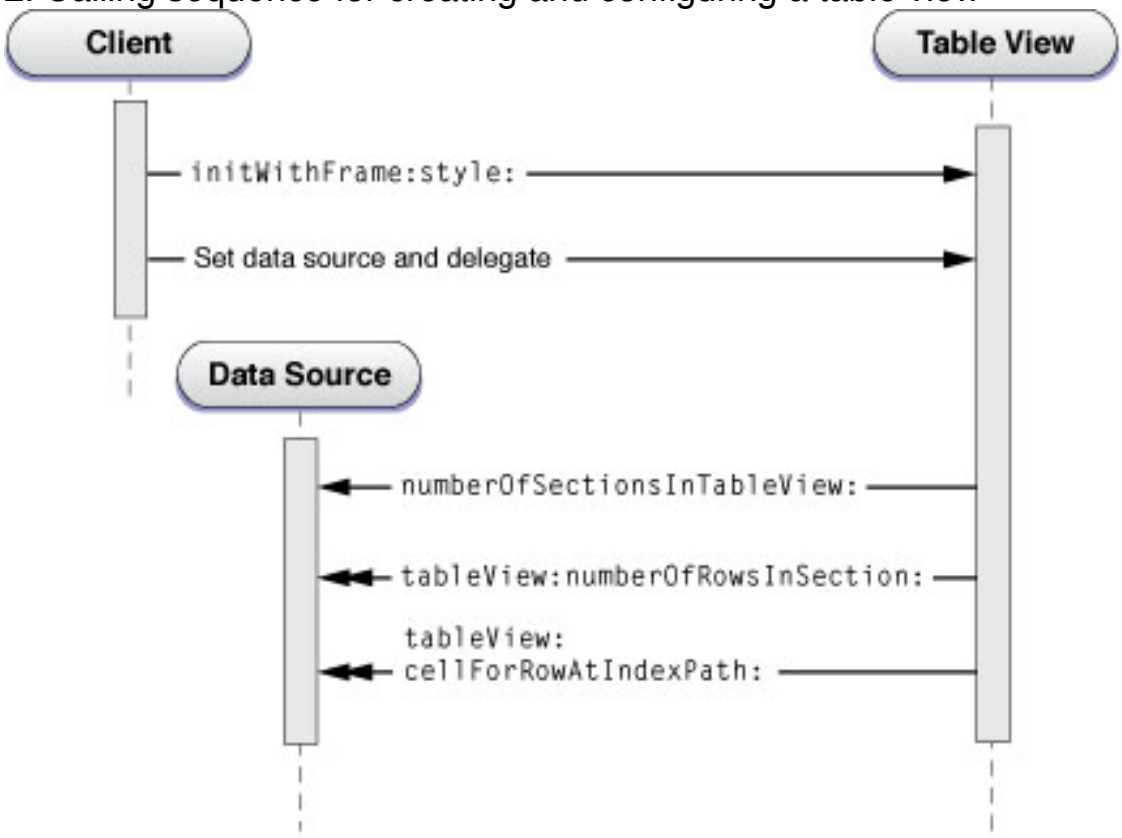
Creating and Configuring a Table View

Basics of Table View Creation
1. To create a table view, several entries in an app must interact the view controller, the table view itself, and the table view's data source and delegate.

2. Calling sequence for creating and configuring a table view



Recommendations for Creating and Configuring Table Views
1. The following approaches are recommended:
=> Use an instance of a subclass of UITableViewController to create and manage a table view.
=> If your app is largely based on table views, select the Master-Detail Application template provided by Xcode when you create your project.
=> For successive table views, you should implement custom UITableViewController objects. You can either load them from a storyboard or create the associated table views programmatically.

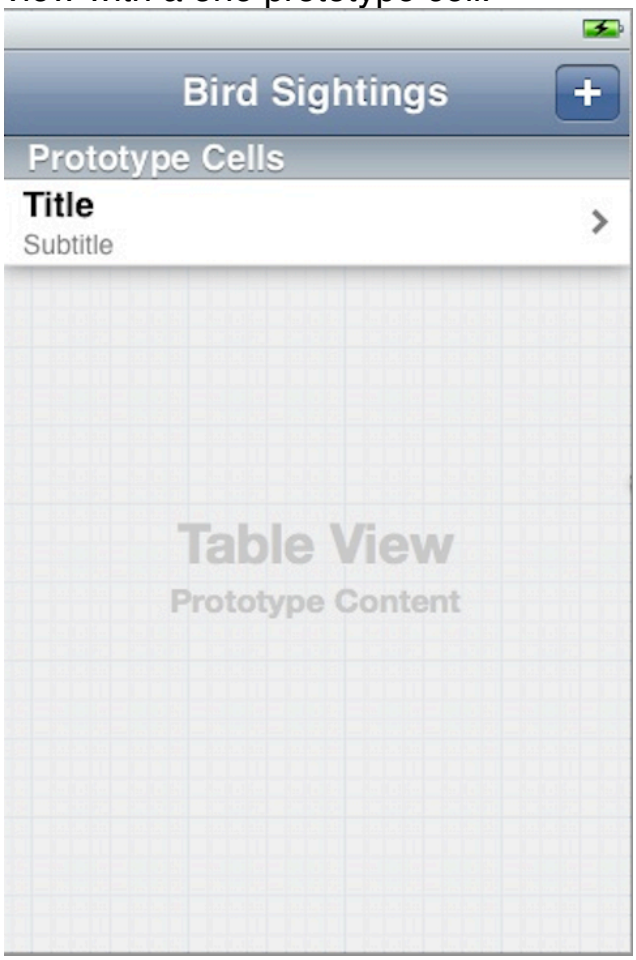
2. If the view to be managed is a composite view in which a table view is one of multiply subviews, you must use a custom subclass of

UIViewController to manage the table view.

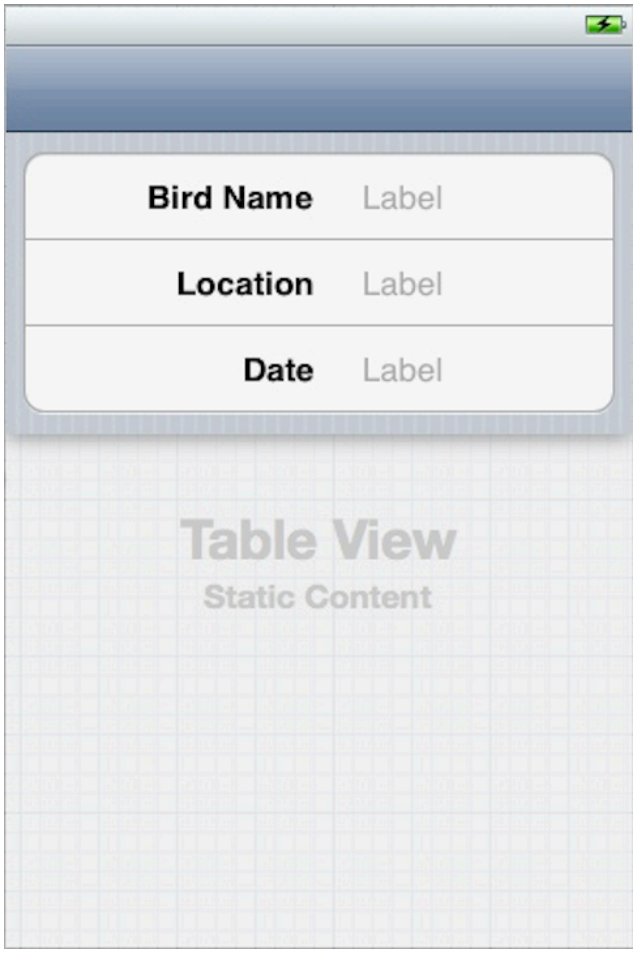
Creating a Table View Using a Storyboard

> Choose the Table View's Display Style

> Choose the Table View's Content Type
1. Storyboards introduce two convenient ways to design a table view's content.
=> **Dynamic prototypes**. Design a prototype cell and then use it as the template for other cells in the table. Figure below shows a plain table view with a one prototype cell.



=> **Static cells**. Use static content to design the overall layout of the table, including the total number of cells. Figure below shows a grouped table view with three static cells.



2. If you're designing a prototype cell, the table view needs a way to identify the prototype when the data source dequeues reusable cells for the table at runtime. In the Table View Cell section of the Attributes inspector, enter an ID in the Identifier text field. By convention, a cell's reuse identifier should describe what the cell contains, such as BirdSightingCell.

> Design the Table View's Rows

> Create Additional Table Views

> Learn More by Creating a Sample App

Creating a Table View Programmatically

> Adopt the Data Source and Delegate Protocols
1. Adopting the data source and delegate protocols

```
@interface RootViewController : UIViewController
<UITableViewDelegate, UITableViewDataSource>
```

```
@property (nonatomic, strong) NSArray *timeZoneNames;
@end
```

> Create and Configure a Table View
1. Creating a table view

```
- (void)loadView
{
    UITableView *tableView = [[UITableView alloc] initWithFrame:
[[UIScreen mainScreen] applicationFrame] style:UITableViewStylePlain];
    tableView.autoresizingMask = UIViewAutoresizingFlexibleHeightl
UIViewAutoresizingFlexibleWidth;
    tableView.delegate = self;
    tableView.dataSource = self;
    [tableView reloadData];

    self.view = tableView;
}
```

Populating a Dynamic Table View with Data
1. The table view repeatedly invokes the tableView:cellForRowAtIndexPath: method to get a cell object for each visible row.

2. Populating a dynamic table view with data

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
    return [regions count];
}

- (NSInteger)tableView:(UITableView *)tableView
numberOfRowsInSection:(NSInteger)section {
    // Number of rows is the number of time zones in the region for the
specified section.
    Region *region = [regions objectAtIndex:section];
    return [region.timeZoneWrappers count];
}



- (NSString *)tableView:(UITableView *)tableView
titleForHeaderInSection:(NSInteger)section {
    // The header for the section is the region name -- get this from the
```

```
region at the section index.
    Region *region = [regions objectAtIndex:section];
    return [region name];
}


- (UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath {
    static NSString *MyIdentifier = @"MyReuseIdentifier";
    UITableViewCell *cell = [tableView
dequeueReusableCellWithIdentifier:MyIdentifier];
    if (cell == nil) {
        cell = [[UITableViewCell alloc]
initWithStyle:UITableViewCellStyleDefault  reuseIdentifier:MyIdentifier];
    }
    Region *region = [regions objectAtIndex:indexPath.section];
    TimeZoneWrapper *timeZoneWrapper = [region.timeZoneWrappers
objectAtIndex:indexPath.row];
    cell.textLabel.text = timeZoneWrapper.localeName;
    return cell;
}
```

Populating a Static Table View With Data

1. Populating a static table view with data

```
- (void)viewDidLoad
{
    [super viewDidLoad];

    BirdSighting *theSighting = self.sighting;
    static NSDateFormatter *formatter = nil;
    if (formatter == nil) {
        formatter = [[NSDateFormatter alloc] init];
        [formatter setDateStyle:NSDateFormatterMediumStyle];
    }
    if (theSighting) {
        self.birdNameLabel.text = theSighting.name;
        self.locationLabel.text = theSighting.location;
        self.dateLabel.text = [formatter stringFromDate:
(NSDate*)theSighting.date];
    }
}
```

Populating an Indexed List

1. Defining the model-object interface

```
@interface State : NSObject

@property(nonatomic,copy) NSString *name;
@property(nonatomic,copy) NSString *capitol;
@property(nonatomic,copy) NSString *population;
@property NSInteger sectionNumber;
@end
```

2. Loading the table-view data and initializing the model objects

```
- (void)viewDidLoad {
    [super viewDidLoad];
    UILocalizedIndexedCollation *theCollation =
[UILocalizedIndexedCollation currentCollation];
    self.states = [NSMutableArray arrayWithCapacity:1];

    NSString *thePath = [[NSBundle mainBundle]
pathForResource:@"States" ofType:@"plist"];
    NSArray *tempArray;
    NSMutableArray *statesTemp;
    if (thePath && (tempArray = [NSArray
arrayWithContentsOfFile:thePath]) ) {
        statesTemp = [NSMutableArray arrayWithCapacity:1];
        for (NSDictionary *stateDict in tempArray) {
            State *aState = [[State alloc] init];
            aState.name = [stateDict objectForKey:@"Name"];
            aState.population = [stateDict objectForKey:@"Population"];
            aState.capitol = [stateDict objectForKey:@"Capitol"];
            [statesTemp addObject:aState];
        }
    } else  {
        return;
    }
```

3. Preparing the data for the indexed list

```
    // viewDidLoad continued...
    // (1)
    for (State *theState in statesTemp) {
        NSInteger sect = [theCollation sectionForObject:theState
```

```
collationStringSelector:@selector(name)];
        theState.sectionNumber = sect;
    }
    // (2)
    NSInteger highSection = [[theCollation sectionTitles] count];
    NSMutableArray *sectionArrays = [NSMutableArray
arrayWithCapacity:highSection];
    for (int i = 0; i < highSection; i++) {
        NSMutableArray *sectionArray = [NSMutableArray
arrayWithCapacity:1];
        [sectionArrays addObject:sectionArray];
    }
    // (3)
    for (State *theState in statesTemp) {
        [(NSMutableArray *)[sectionArrays
objectAtIndex:theState.sectionNumber] addObject:theState];
    }
    // (4)
    for (NSMutableArray *sectionArray in sectionArrays) {
        NSArray *sortedSection = [theCollation
sortedArrayFromArray:sectionArray
            collationStringSelector:@selector(name)];
        [self.states addObject:sortedSection];
    }
} // end of viewDidLoad
```

4. Providing section-index data to the table view

```
- (NSArray *)sectionIndexTitlesForTableView:(UITableView *)tableView {
    return [[UILocalizedIndexedCollation currentCollation]
sectionIndexTitles];
}

- (NSString *)tableView:(UITableView *)tableView
titleForHeaderInSection:(NSInteger)section {
    if ([[self.states objectAtIndex:section] count] > 0) {
        return [[[UILocalizedIndexedCollation currentCollation]
sectionTitles] objectAtIndex:section];
    }
    return nil;
}
```

```
- (NSInteger)tableView:(UITableView *)tableView
sectionForSectionIndexTitle:(NSString *)title atIndex:(NSInteger)index
{
    return [[UILocalizedIndexedCollation currentCollation]
sectionForSectionIndexTitleAtIndex:index];
}
```

5. Populating the rows of an indexed list

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
    return [self.states count];
}

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

- (UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath {
    static NSString *CellIdentifier = @"StateCell";
    UITableViewCell *cell;
    cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier];
    if (cell == nil) {
        cell = [[UITableViewCell alloc]
initWithStyle:UITableViewCellStyleDefault reuseIdentifier:CellIdentifier];
    }
    State *stateObj = [[self.states objectAtIndex:indexPath.section]
objectAtIndex:indexPath.row];
    cell.textLabel.text = stateObj.name;
    return cell;
}
```

Optional Table View Configurations

> Add a Custom Title

1. Adding a title to the table view

```
- (void)loadView
{
    CGRect titleRect = CGRectMake(0, 0, 300, 40);
    UILabel *tableTitle = [[UILabel alloc] initWithFrame:titleRect];
    tableTitle.textColor = [UIColor blueColor];
```

```
    tableTitle.backgroundColor = [self.tableView backgroundColor];
    tableTitle.opaque = YES;
    tableTitle.font = [UIFont boldSystemFontOfSize:18];
    tableTitle.text = [curTrail objectForKey:@"Name"];
    self.tableView.tableHeaderView = tableTitle;
    [self.tableView reloadData];
}
```

> Provide a Section Title

1. Returning a title for a section

```
- (NSString *)tableView:(UITableView *)tableView
titleForHeaderInSection:(NSInteger)section {
    // Returns section title based on physical state: [solid, liquid, gas,
artificial]
    return [[[PeriodicElements sharedPeriodicElements]
elementPhysicalStatesArray] objectAtIndex:section];
}
```

> Indent a Row

1. Custom indentation of a row

```
- (NSInteger)tableView:(UITableView *)tableView
indentationLevelForRowAtIndexPath:(NSIndexPath *)indexPath {
    if ( indexPath.section==TRAIL_MAP_SECTION &&
indexPath.row==0 ) {
        return 2;
    }
    return 1;
}
```

> Vary a Row’s Height

1. Varying row height

```
- (CGFloat)tableView:(UITableView *)tableView
heightForRowAtIndexPath:(NSIndexPath *)indexPath
{
    CGFloat result;

    switch ([indexPath row])
    {
        case 0:
        {
            result = kUIRowHeight;
```

```
            break;
        }
        case 1:
        {
            result = kUIRowLabelHeight;
            break;
        }
    }
    return result;
}
```

> Customize Cells