MEMORIAL UNIVERSITY Department of Computer Science

COMP 4768: Software Development for Mobile Devices

Table View

Yuanzhu CHEN
http://www.cs.mun.ca/~yzchen/
yzchen@mun.ca

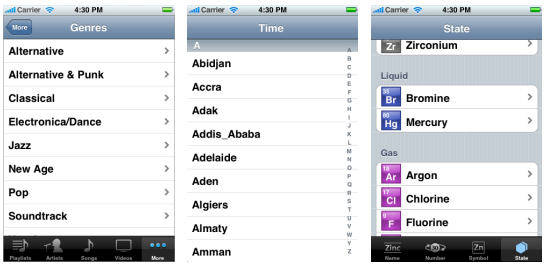
Winter 2016

Today's topic

- Table view generics
- Creation
- Managing selections
- Insertion and deletion
- Reordering

Table View

- Table views are commonly found in iOS applications
- A table presents a list of items that might be divided into sections, often to
  - let users navigate through hierarchically structured data
  - present an indexed list of items
  - display detailed information and controls in visually distinct groups
  - provide options
- Scrollable vertically



Characteristics of Table Views

- A table view may have a header and/or footer
- It consists of rows in sections
  - Each section may have its own header and/or footer
  - Rows are thus identified with a two-tier index path
- Visible rows of a table view are composed of cells
  - which displays images, texts, and other contents
  - Cells can also have accessory views, often as controls. They are
- `UITableViewCellAccessoryNone`,
- `UITableViewCellAccessoryDisclosureIndicator`,
- `UITableViewCellAccessoryDetailDisclosureButton`,
- `UITableViewCellAccessoryCheckmark`,
- `UITableViewCellAccessoryDetailButton`

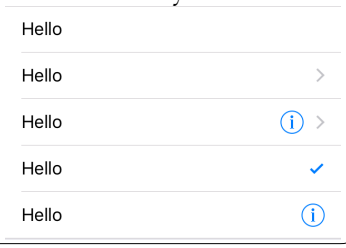
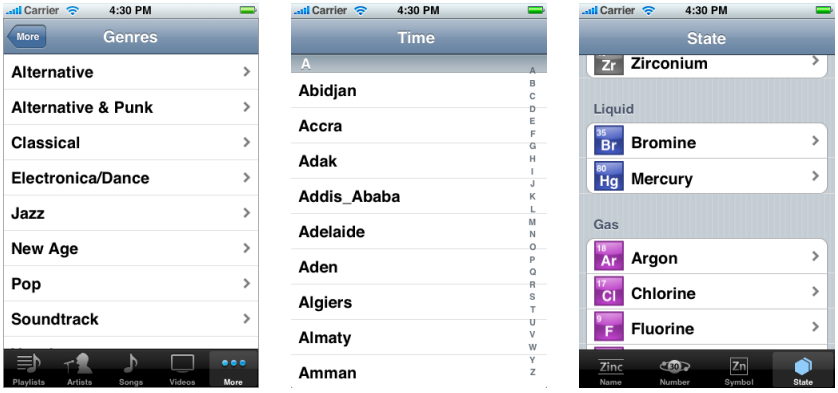


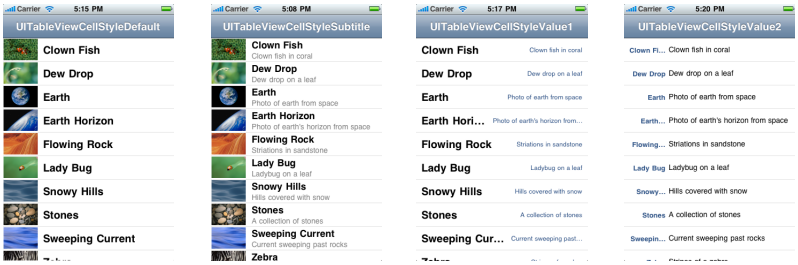
Table View Styles

- Plain – one or more sections
  - It can have an indexed list on the right for sections
- Grouped – sections are separated in rounded rectangles



Styles of Table-View Cells

- Standard styles are
  - `UITableViewCellStyleDefault`
  - `UITableViewCellStyleSubtitle`
  - `UITableViewCellStyleValue1`
  - `UITableViewCellStyleValue2`



- Cells can also be customized

Table View API at a Glance

- A table view is an instance of `UITabiewView` class, a subclass of `UIScrollView`. This class allows you to
  - configure its appearance, e.g. default row height and header view
  - access to a specific row
  - manage selections
  - scroll to a position programmatically
  - insert, delete, or reorder rows
- A `UITableView` object must have a
  - data source – as data model in MVC, of `<UITableViewDataSource>`
  - delegate – manager of appearance and behavior, of `<UITableViewDelegate>`
  - These data source and delegate are often the same object, and is frequently an instance of custom subclass of `UITablewViewController`

Data Source and Delegate

- The `dataSource` property conforms to `<UITableViewData-Source>`, which tells the table view
  - how many sections there are
  - how many rows there are in a given section
  - what the titles of the header and footer are for a section
  - the content of a given cell
- The `delegate` property conforms to `<UITableView-Delegate>`,
  - which notifies the table view of a user
    - set visual traits of a row
    - selecting and deselecting a row
    - adding, deleting, and moving a row
  - and manages the accessory views

Table Views and Data Model

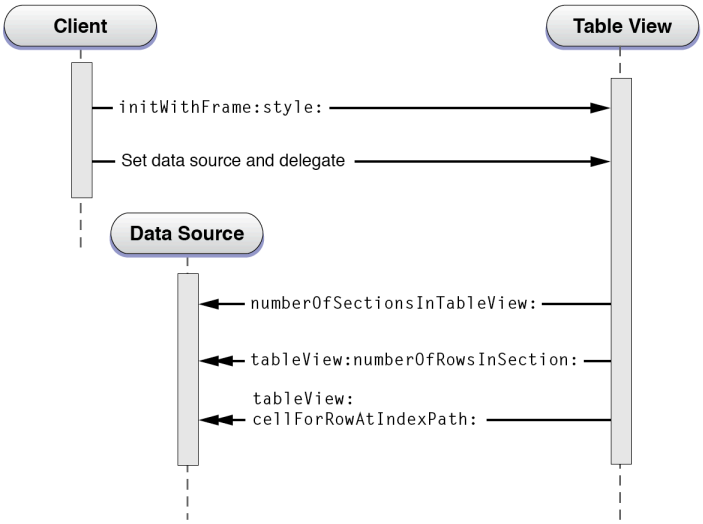
- Rows of a table view are typically backed by collection objects in the application's data model, usually arrays
  - The array contains strings or other elements that the table view can use when displaying row content
  - When you create a table view, the table view immediately queries its data source for
    - its dimensions, i.e. number of sections and number of rows per section
    - content for each **visible** row
- In many methods defined for the table view and its data source and delegate, the table view passes in an *index path* to identify the section and row of current operation focus
  - The index path is an instance of the `NSIndexPath` class, where its `section` and `row` properties are of interest here

Mapping Levels of Data Model to Table View

Table-View Controller

- Although you could manage a table view using a direct subclass of `UIViewController`, you save yourself a lot of work if you instead subclass `UITableViewController`
  - The `UITableViewController` class takes care of many details you would have to implement otherwise
- To create a table-view controller, you allocate memory space and init it with a style in either `UITableViewStylePlain` or `UITableViewStyleGrouped`
- Whether loading a table view from a nib file or programmatically, the table-view controller is the data source and delegate of the table view
- When the table view is about to appear for the first time, the controller sends it `reloadData`, prompting it to request data from its data source
  - The data source tells the the table view how many sections and rows there are and gives it the data to display in each row
- Other tasks of `UITableViewController` include
  - clearing selections when the table is about to be displayed, and
  - flashing the scroll indicator when the table finishes displaying

Basics of Table View Creation



Creating a Table View Programmatically

- A preferred approach is to subclass UITableViewController, which takes care of data source and delegate by itself
- If you choose to directly subclass UIViewController, conform to protocols

```
@interface MyViewController : UIViewController <UITableViewDelegate,
                UITableViewDataSource> {
     NSArray *timeZoneNames;
}
@property (nonatomic, retain) NSArray *timeZoneNames;
@end
```

- You can create the table view when initializing the view controller, e.g.

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

Populating Table View with Data

- After a table view is created, it receives a reloadData message, which tells it to start querying the data source and delegate for the information needed to display sections and rows, via
  - `numberOfSectionsInTableView:`
  - `tableView:numberOfRowsInSection:`
  - `tableView:cellForRowAtIndexPath:`
  - `tableView:titleForHeaderInSection:`
  - `tableView:titleForFooterInSection:`

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
     return [regions count];
}
- (NSInteger)tableView:(UITableView *)tableView
                numberOfRowsInSection:(NSInteger)section {
     Region *region = [regions objectAtIndex:section];
     return [region.timeZoneWrappers count];
}
- (NSString *)tableView:(UITableView *)tableView
                titleForHeaderInSection:(NSInteger)section {
     Region *region = [regions objectAtIndex:section];
     return [region name];
}
```

Reusing Table View Cells

- The data source in its implementation of `tableView:cellForRowAtIndexPath:` returns a configured cell object that the table view can use to draw a row
- For performance reasons, the data source tries to reuse cells as much as possible
  - It first asks the table view for a specific reusable cell object by sending it a message of `dequeueReusableCellWithIdentifier:`
  - If no such object exists, the data source creates it, assigning it a reuse identifier

```
- (UITableViewCell *)tableView:(UITableView *)tableView
                cellForRowAtIndexPath:(NSIndexPath *)indexPath {
     static NSString *MyIdentifier = @"MyIdentifier";
     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;
}
```

Managing Selections

- When a user taps a row of a table view, a response can be
  - another table view slides into place
  - row displays a checkmark
  - other change in user interface or backend data
- There are a few human-interface guidelines for table views
  - You should never use selection to indicate state
    - Instead, use checkmarks
  - When a user selects a cell, you should respond by
    - deselecting the item via `deselectRowAtIndexPath:Animated:`, and
    - performing any appropriate action, e.g. presenting a detailed view
  - If you respond by pushing a new view controller onto the navigation stack, you should deselect the cell (with animation) when the view controller is popped off the stack

Example: State with Checkmarks

- When the row is tapped, the delegate is messaged with `tableView:didSelectRowAtIndexPath:`
  - You need to implement this method as you wish the app to respond
- Here is an example of toggling the checkmark of the tapped row

```
-(void)tableView:(UITableView *)tableView
         didSelectRowAtIndexPath:(NSIndexPath*)indexPath
{
    UITableViewCell *currentCell = [tableView cellForRowAtIndexPath:indexPath];
    [tableView deselectRowAtIndexPath:indexPath animated:YES];
    if (currentCell.accessoryType == UITableViewCellAccessoryNone)
        currentCell.accessoryType = UITableViewCellAccessoryCheckmark;
    else
        currentCell.accessoryType = UITableViewCellAccessoryNone;
}
```

- Note that if a row has a "detail disclosure button", tapping the control sends a `tableView:accessoryButtonTappedForRowWithIndexPath:` message to the delegate

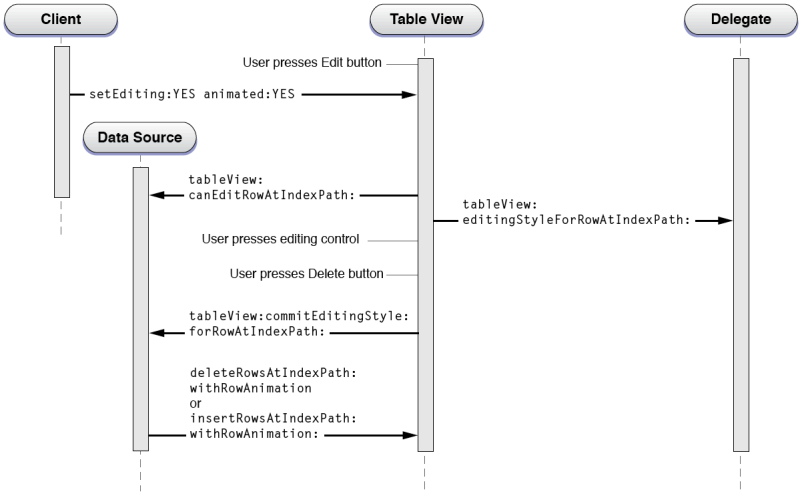
Insertion and Deletion

- A table view has a normal (selection) mode and an editing mode
- When in editing mode, the editing controls (in the left side of the row) allows the user to insert and delete rows in the table view

  - In addition, the user is also allowed to reorder the rows
- A table view goes into editing mode when it receives the `setEditing:animated:` message.
  - Typically, the message originates as an action message sent when the user taps the **Edit** button in the navigation bar
  - In editing mode, a table view displays editing and reordering controls that its delegate has assigned to each row.
    - The delegate assigns the controls as a result of returning the editing style for a row in the `tableView:editingStyleForRowAtIndexPath:` method

Calling Sequence

Calling sequence for inserting or deleting in a table view



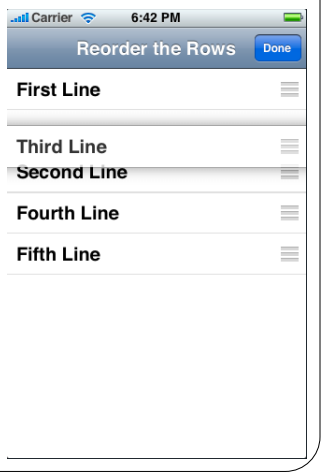
Example of Insertion and Deletion

- To perform the actual operations, you need to implement the `tableView:commitEditingStyle:forRowAtIndexPath:` method of the data source

```
- (void)tableView:(UITableView *)tableView
        commitEditingStyle:(UITableViewCellEditingStyle)editingStyle
         forRowAtIndexPath:(NSIndexPath *)indexPath
{
    if (editingStyle == UITableViewCellEditingStyleDelete)
    {
        [timeZoneNames removeObjectAtIndex:indexPath.row];
        [tableView deleteRowsAtIndexPaths:[NSArray arrayWithObject:indexPath]
            withRowAnimation:UITableViewRowAnimationFade];
    }
    else if (editingStyle == UITableViewCellEditingStyleInsert)
    {
        NSString *stringToAdd = @"New/World!";
        [timeZoneNames insertObject:stringToAdd atIndex:indexPath.row];
        [tableView rtRowsAtIndexPaths:[NSArray arrayWithObject:indexPath]
            withRowAnimation:UITableViewRowAnimationBottom];
    }
}
```

Managing Reordering of Rows

- When in editing mode, the table view can also display the reordering controls for the user to rearrange the rows
- When the user drags a reordering control, a series of messages are sent to its data source and delegate
- You will need to implement these methods to control if and how rows can be reordered



Calling Sequence for Reordering

Calling sequence for reordering rows in a table view

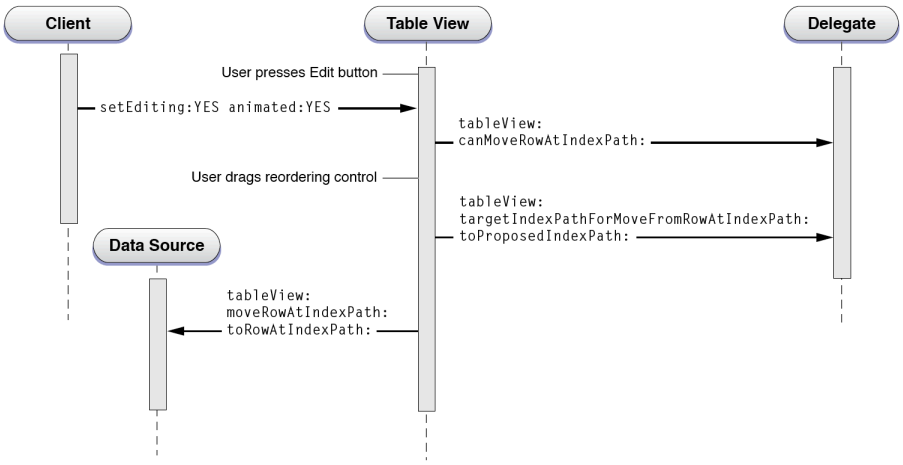
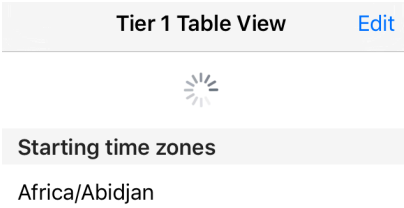


Table View Refresh

- A `UIRefreshControl` object provides a standard control that can be used to initiate the refreshing of a table view's contents.
- The table view controller handles the work of adding the control to the table's visual appearance and managing the display of that control in response to appropriate user gestures.

- Assigning a refresh control to a table view controller's `refreshControl` property
- Configure the target and action of the control



Readings

- *TableView Programming Guide for iOS*, an Apple document