Lab – Tables

OVERVIEW

Date Due: Thursday, March 11th (2 weeks) by 11:59 p.m. EST. ✧ **Value:** 40 points

Features

The ToDo application is being extended to display two different editable views. The date list view presents events ordered by date, and sectioned (grouped) by day. Priority list view presents events ordered by priority. Each list provides editing features. Users can switch between views using a segmented control (or tab bar if you prefer).

The date list view groups events with the same day into sections. Events are deletable (but not reorderable) in this view. Each section header displays the due day of its member events. Since the day is displayed in the header, we can avoid information duplication and clean up the UI. To that end, we will display just the title and time on a single line.

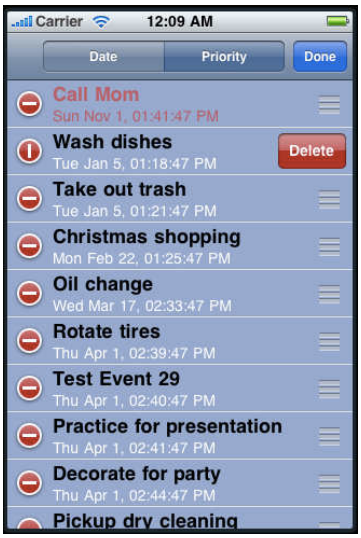
The priority list view displays unfinished events ordered by a priority. It allows events to be deleted or reordered so users can quickly prioritize. To help users quickly locate past due events, the priority list view will display those items in red.

Resources

Starter resources can be found here: http://www.cs.umd.edu/class/spring2010/cmsc498i/files/lab4_resources.zip. `NSDateAdditions.[hm]` provides functionality needed to display dates, and do comparisons for grouping purposes. `ToDoCell.[hm]` provides stubbed out methods you can use as a staring point for display. `ToDoEvent.[hm]` is the model object, and implements everything you should need. Finally, `SampleEvents.plist` is a property list containing the events you should load and display.

Notes

Instead of providing pre-written steps to follow, labs will now be more free form. The notes that follow are suggestions to issues you need to solve and code you need to implement.



NOTES

Suggested Approach

Instead of being given a numbered list of steps, a suggested order to work on parts is being provided as a guide.

Begin by creating your project (Using a "Navigation-based application" is recommended). Next, create your user interface. Define outlets and actions, and wire them up. Finally, get to work on the code. Begin by wiring up the code to switch between your two blank table views. This will require you to begin thinking about your architecture and set up infrastructure. Next, get the priority list view to display events (worry about editing later). This will get your application loading data, and take you through using a `UITableViewCell`. After getting your first list displaying data, work on getting the date list view to display it's data. The biggest challenge here will be creating and working with sectioned data. Finally, once you have all of this done, add editing capability to the application, test and submit.

Coding Guidelines and Style

An important part of writing clean and readable code is following procedures and conventions used by others. Overtime you will develop your own practices and style. Most Cocoa developers subscribe to a standard set of conventions. Please read:

> CodingGuidelines.pdf – p. 8 - 23, p. 33 - 35
>
> CocoaFundamentals.pdf – "Cocoa API Conventions", p.112-114.

Bundle Resource

Xcode allows you to bundle resources (images, strings files, plists) with your application. `NSBundle` represents a grouping of codeand resources in a directory. There are many types of bundles, and your application itself stored on the filesystem in a bundle. `NSBundle's` "main bundle" gives you access your application's bundle and the resources contained within. For example, to find the SampleEvents.plist that you packaged with your appication, you use

```
[[NSBundle mainBundle] pathForResource:@"SampleEvents" ofType:@"plist"]
```

Sectioning Events

Creating and dealing with sectioned data is one of the more difficult parts of this assignment.

Probably the most straight forward approach to sectioning your data involves sorting events and then grouping the events into sections. When determening group membership, make sure to use the `NSDate` helper methods in the `NSDateAdditions` source. Storage of section data can be done many ways. You could consider an `NSArray` of `NSArrays` for each group. Or you could keep two arrays around with the first array being the events in sorted order and the second being an array of section index information. For example, each section could be represented by an `NSIndexSet` of the indexes in that section. Or, you could store `NSRange's` (by using `+[NSValue valueWithRange:]`).

NOTES

Table Editing Interface

`UITableView` provides API to make table editing UI really easy, just call `-setEditing:animated:`. `UIViewController` also makes life easy. To display the edit button in the navigation bar ("navigation-based" apps come with one) you write the following: `[[self navigationItem] setRightBarButtonItem:[self editButtonItem];`. This accesses `UIViewController's` standard edit button and attaches it to the right side of the navigation item. When the top level navigation controller installs your `UIViewController` it will use your navigation item to configure its navigation bar. The standard edit button is pre-configured to call the `-setEditing:animated:` on your `UIViewController` subclass. You will need to override this to make your table start and stop editing. Make sure to call super's implementation. Experiment with not calling `[super setEditing:editing animated:animated]` and see if you can figure out what functionality this gains you. This is an example where subclassing a method allows you to customize behavior. However, you need to understand what behavior you are overriding and decide whether or not you are adding to, or replacing that behavior. In this case, we are adding to the behavior.

Table Editing

During this lab you will become familiar with the editing related table data source methods.

The priority list view supports deletion and reordering. `UITableView` does not reorder the data itself and instead tells its data source to take care of updating the model through a callback. To simplify reordering, assume that events have unique, real number, priorities (you have to maintain this of course) in the range (0, 5). Also assume that events in the supplied plist have unique priorities. Given that, reordering is as simple as computing a new unique priority value based on the events new neighbors.

The date list view supports only deletion. You may need to tell your table (using a data source method -- see if you can figure out which one on your own) that reordering is not supported if you are using one controller for both the date and priority list display.

Reordering is similar to deletion in that `UITableView` tells its data source to update the model. Reordering and deletion differ in one regard. When dealing with reordering, `UITableView` already knows where it is supposed to drop the row for you, so everything is specified. With deletion, you need to tell the table how to make the deleted row disappear using `-deleteRowsAtIndexPaths:withRowAnimation:`, `-deleteSections:withRowAnimation:`. Be careful when deciding which delete method to use. One of these is for making rows disappear, one is for making a section disappear when no more rows would remain in the section.

NOTES

Cell Display

When the data associated with a cell changes, it is tempting to immediately update the display (labels, fonts, etc) immediately. However, it can often be the case that multiple changes can happen quckly, or multiple properties can be involved in configuring the display state. To improve performance, instead of updating UI immediately when a property changes, you should schedule the update by calling `-[UIView setNeedsLayout]`. By doing so, you schedule your subclass's `-layoutSubviews` method to be invoked before display. You can do all your display related updates there.

When positioning content in a `UITableViewCell`, remember that default labels are subviews of the cell's `-contentView`. If you need to (you may not need to) add your own subviews, they should be subviews of the `contentView`. Finally, keep in mind that when the editing controls display, they will shrink the `contentView`, so don't assume the `contentView` is always going to be the same width.