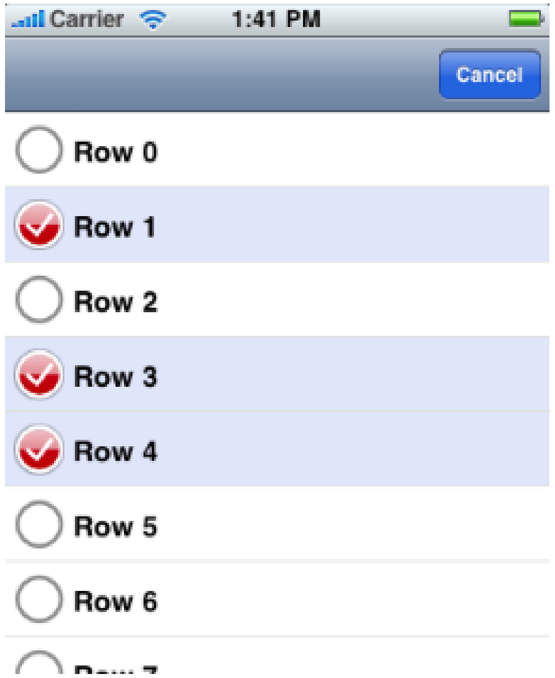
Multiple row selection and editing in a UITableView

By default, UITableView only supports single-row selection. In this post, I'll show you how to implement multi-row selection, similar to the message edit interface in Mail.

Introduction

The target behavior for this post is an editing mode for a UITableView which allows the selection of multiple rows and presents a button to perform an action on the selection rows.

The following is a screenshot of the sample application running:

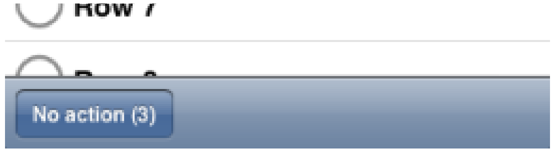


Search

Search

Archive

▶ 2011 (10)
▶ 2010 (41)
▼ 2009 (56)
▶ December 2009 (5)
▶ November 2009 (5)
▶ October 2009 (5)
▶ September 2009 (6)
▶ August 2009 (5)
▶ July 2009 (5)
▶ June 2009 (4)
▶ May 2009 (4)
▶ April 2009 (4)
▶ March 2009 (5)
▶ February 2009 (4)
▼ January 2009 (4)
- Multiple virtual pages in a UIScrollView with just...
- Demystifying NSApplication by recreating it
- Multiple row selection and editing in a UITableVie...



When not editing, neither the column of circles and check marks nor the bottom toolbar is visible. When the "Edit" button is clicked (located in place of the cancel button above), the "Edit" button is replaced by the "Cancel" button and the circles and check marks column and the bottom toolbar animate in.

Requirements for the implementation

`UITableView` does not support multiple selection. We will use the method `tableView:didSelectRowAtIndexPath:` to detect touches in rows but the selected state will need to be stored separately (we cannot rely on the `UITableView`'s selection).

We will also need a background view for displaying the selection color and a `UIImageView` for displaying the not-selected/selected indicator. Since the `UIImageView` will be hidden while not editing and the label for the row needs to move left or right when it is shown or hidden, we will also need to implement some form of layout for the `UITableViewCell`.

Other required behaviors include switching the "Edit"/"Cancel" buttons between modes, showing/hiding the toolbar at the bottom and tracking the number of selected rows to display in the button in the toolbar.

Implementation

The implementation begins with Apple's default "Navigation-based Application" template.

I then changed the `RootViewController` to be a subclass of the `GenericTableViewController` implementation that I presented in my previous Heterogeneous cells in a UITableViewController post. In that post, this class was presented to aid handling of *different* cell types in one table. I use it again here with only one cell type because in this case, the `CellController` provides a convenient object in which to store the "selected" state for each row and allows me to keep each file smaller and narrower in focus because it keeps "row" behavior out of the table controller.

Toolbar

The first addition I made was that of the toolbar. This is initially hidden but needs to animate onto the screen when the edit mode is entered.

The toolbar is constructed in the `viewDidLoad` implementation:

- Serving an NSManagedObjectContext over an NSConnec...

▶ 2008 (47)

Categories

- AppKit (38)
- C++ (5)
- Cocoa Touch (32)
- code management (3)
- Core Data (10)
- CoreAnimation (7)
- CoreAudio (5)
- CoreGraphics (8)
- Foundation (45)
- fun hacks (12)
- graphics (6)
- Lion (1)
- numerical methods (3)
- object-oriented design (17)
- Objective-C (38)
- performance tests (7)
- self promotion (6)
- Snow Leopard (6)
- Standard C (17)
- testing (6)
- UIKit (26)
- WebKit (3)
- Xcode (18)

RSS/Atom and Twitter Feeds

Subscribe to this blog

cocoawithlove on Twitter

```
actionToolbar = [[UIToolbar alloc] initWithFrame:CGRectMake(0, 416, 320, 44)];
actionButton =
    [[[UIBarButtonItem alloc]
        initWithTitle:@"No Action"
        style:UIBarButtonItemStyleBordered
        target:self
        action:@selector(noAction:)]
    autorelease];
[actionToolbar setItems:[NSArray arrayWithObject:actionButton]];
```

but cannot be easily added to the view hierarchy at this time. Instead, we wait until the `viewDidAppear:` method is invoked and add it as a child of the table's parent (the `UINavigationController`'s content frame):

```
- (void)viewDidAppear:(BOOL)animated
{
    [self.view.superview addSubview:actionToolbar];
}
```

The initial location of the toolbar is below the bottom of the screen, so when editing begins, we need to move it up onto the screen. When editing ends, it is moved back again. This frame animation occurs in the `showActionToolbar:` method.

Edit Mode

Standard edit modes for `UITableView`s are started by calling `setEditing:animated:` on the `UITableView`. We are not going to use any of the standard `UITableViewCellEditingStyle`s, so invoking this method is not strictly required but it will propagate a notification to the `UITableViewCell`s and allow us to query the state at a later time so we will use it anyway.

The `edit:` and `cancel:` methods switch us into and out of "Edit" mode respectively.

```
- (void)edit:(id)sender
{
    [self showActionToolbar:YES];

    UIBarButtonItem *cancelButton =
        [[[UIBarButtonItem alloc]
            initWithTitle:@"Cancel"
            style:UIBarButtonItemStyleDone
            target:self
            action:@selector(cancel:)]
        autorelease];
    [self.navigationItem setRightBarButtonItem:cancelButton animated:NO];
    [self updateSelectionCount];

    [self.tableView setEditing:YES animated:YES];
}

- (void)cancel:(id)sender
{
    [self showActionToolbar:NO];

    UIBarButtonItem *editButton =
        [[[UIBarButtonItem alloc]
            initWithTitle:@"Edit"
            style:UIBarButtonItemStylePlain
            target:self
            action:@selector(edit:)]
        autorelease];
    [self.navigationItem setRightBarButtonItem:editButton animated:NO];

    NSInteger row = 0;
    for (MultiSelectCellController *cellController in
        [tableGroups objectAtIndex:0])
    {
        [cellController clearSelectionForTableView:self.tableView
            indexPath:[NSIndexPath indexPathForRow:row inSection:0]];
        row++;
    }

    [self.tableView setEditing:NO animated:YES];
}
```

Here you can see the "Edit"/"Cancel" buttons being swapped, the toolbar being shown/hidden and `setEditing:animated:` being invoked. I also implement `tableView:canEditRowAtIndexPath:` to always return yes, since all rows may be edited in this table.

Showing/hiding the check mark column

When `setEditing:animated:` is invoked on the table, the table in turn invokes the `setEditing:animated:` on all visible `UITableViewCell`s, allowing each row to update for editing.

In response to this, we need to show/hide the check mark column. We handle this in a `UITableViewCell` subclass where the `setEditing:animated:` is implemented to call `setNeedsLayout` and the `layoutSubviews` method is overridden to handle different layouts for the "Edit" an "Not Editing" modes.

When editing, the cell's `contentView` is shifted to the right, otherwise it is layed out flush against the left side. This is all we'll need to display the extra column because the check mark column is always present in the cell. Outside of editing mode, it is layed out off the left of screen (so you can't see it). When the `contentView` is shifted right by 35 pixels during editing, the check mark column (which is located at the `contentView`'s origin minus 35 pixels horizontally) becomes visible.

```
- (void)setEditing:(BOOL)editing animated:(BOOL)animated
{
    [self setNeedsLayout];
}

- (void)layoutSubviews
{
    [UIView beginAnimations:nil context:nil];
    [UIView setAnimationBeginsFromCurrentState:YES];

    [super layoutSubviews];

    if (((UITableView *)self.superview).isEditing)
    {
        CGRect contentFrame = self.contentView.frame;
        contentFrame.origin.x = EDITING_HORIZONTAL_OFFSET;
        self.contentView.frame = contentFrame;
    }
    else
    {
        CGRect contentFrame = self.contentView.frame;
        contentFrame.origin.x = 0;
        self.contentView.frame = contentFrame;
    }

    [UIView commitAnimations];
}
```

The `setEditing:animated:` implementation ensures that re-layout occurs every time edit mode is entered/exited.

Notice that no custom drawing happens here in the `UITableViewCell`. I've seen many people override `UITableViewCell` for custom drawing but I don't think it's a good idea. The `UITableViewCell` is really just a layout container and that should be the only way you use it. Custom drawing should go in the `backgroundView`, `selectionView` or `contentView` that the `UITableViewCell` contains.

Drawing the cell

I use a `UILabel` for the text rather than setting the `text` property of the cell because it makes it easier to get a transparent background for the text (which I'll need to see the blue selection color).

I set `cell.selectionStyle = UITableViewCellSelectionStyleNone` because I don't want to use the standard selection view at all (it is limited to single rows). Instead, I achieve a selection color by creating a `backgroundView` for the cell and setting its background color to white or pale blue as appropriate.

The selection indicator is just a `UIImageView`. As previously indicated, it is layed out 35 pixels left of the `contentView` which places it offscreen. When the `contentView` is shifted right during editing, it will become visible.

The only other important behavior is that the `CellController` must invoke `updateSelectionCount` on the `RootViewController` when selected/deselected so that the selection count can be updated when the selection changes. I implement this in a lazy fashion by recounting all selected rows — you should probably implement this in a more efficient fashion.

Conclusion

You can download the complete MultiRowSelect Xcode 3.1 project (40kB).

The final result is a few hundred lines of code. This is not a giant mountain of code by any means but still a considerable volume given how simple "multi-row selection" might seem as a description. I think this serves to show that user-interface implementations can be very time consuming when the desired functionality is not provided by the default libraries.

None of the code is particularly complex but it still involves a lot of coordination between the table, table controller and cell so I hope that this sample implementation simplifies the task for anyone else who needs to implement it in the future.

Share this post:

Posted by Matt Gallagher
Saturday, January 10, 2009
Filed in categories: Cocoa Touch, UIKit

Scott Little

Thanks for this example Matt, it is really cool.

I do have a question though. ;-)

Why do you create the actionToolbar and button upon loading of the TableView and not lazily when it is required? It could be created during the showActionToolbar: method for example. Is this just because it's a simple example? Would you do this in an app that may have some memory issues?

Thanks again.
01/12/2009, 05:16:55 – Like – Reply

Matt Gallagher
You are correct, lazy construction would have been better from a design perspective (since it would localize the toolbar code better). I doubt the memory for a simple toolbar would ever warrant careful management so you shouldn't worry about that.

When I've used this code in production, I construct the toolbar as part of the view's XIB file -- which I didn't want to do in this example since it is a bit tricky: you need to wrap the table in a parent view which means you must use a UIViewController as the top level instead of a UITableViewController and handle the table stuff yourself. Instead, the toolbar construction code has been "bolted on" to the Navigation-based Application template -- which is probably why it seems so out of place.
01/12/2009, 06:10:17 – Like – Reply
Liked by Guest

benwulf
I want to point out that the part of the tutorial which mentions that the UITableView selection cannot be relied on due to its monoselection nature is correct but even when you need just a single selection you still cannot rely on it. The UI Guidelines for the iphone specify that the selection needs to be immediately removed and a visual navigation or state changed performed (for instance add a check mark) this is not really just a guideline as I had a application refused because I was leaving the selection on and using it to store the information of which row was selected in a view until the view was dismissed when selecting an action to perform. So even for a single selection code similar to this one is needed although selection can be shown more easily with the checkmark accessory type
01/12/2009, 23:24:34 – Like – Reply

palav
Hi Benwulf,

As mentioned in article http://developer.apple.com/iphone/library/documentation/UserExperience/Conceptual/TableView_iPhone/TableViewStyles/TableViewStyles.html, you can use checkmark to multiple row selection. It is strange that your application was rejected bcoz of same reason...
05/31/2009, 01:36:50 – Like – Reply

Benoit Cerrina

Hello,
there was a misunderstanding regarding the rejection I had and the content of my comment.
My app was rejected for using the selection (blue highlight) as is. This was even when only selecting a single row, and that was my answer to the part of the article about multi row selection.
I also mentioned in my comment that the proper way to handle selection (single or multiple) is to remove the highlight immediately then to use a check mark (whether a standard one in the accesorry view or a custom as you did).

Benwulf
05/31/2009, 08:44:20 – Like – Reply

Andrew Pennebaker

Well done! So there's no iPhone equivalent to the various Cocoa table cell widgets? Lame, but I suppose Apple's development team didn't want to overdesign.
01/13/2009, 10:21:18 – Like – Reply

Raj

Hi Matt,

One of the most useful posts I have come across on UITableView.

I have bookmarked it on http://www.iphonekicks.com/cocoa/Multiple_row_selection_and_editing_in_a_UITableView
01/14/2009, 09:53:58 – Like – Reply

Guest

Hi,

A great tutorial. Thanks for the source. You seem to be the ONLY one who has successfully found a work-around.

I have a question though. I am having trouble implementing your sample source classes into my entire App. I am wondering if this same functionality is workable for a table view that is NOT in 'editing' mode. I am a n00b and also, don't have a "genericTabl eViewController" and a single sub-class of UITableViewController with a standard implementation. Having a hard time trying to use this code effectively.

Any help would be appreciated. Thanks!
03/23/2009, 23:43:39 – Like – Reply

Matt Gallagher

Yes, you can do this without "editing" set.

No, you don't need to use the GenericTableViewController either. The only reason I did was because the CellControllers provided a convenient place to store the "selected" value for each row.

The key steps you will need are setting your own "row X is selected" values somewhere when tableView:didSelectRowAtIndexPath: is invoked and drawing the backgroundView of your table view cells to reflect this value.

03/23/2009, 23:52:38 – Like – Reply

Guest

Thanks!

Decided to store an array of NSNumber (corresponding "Row") objects in the appDelegate.

I see a few buggy behaviors.

If I call -reloadData after making selection (and storing row number in delegate.storedArray), the custom cells are reloaded and not responding to calls within -cellForRowAtIndexPath to 'showAsSelected'). For some reason?

If I do not call -reloadData, I get the representation on screen (for available visual rows) that appear to be multiple persistent selections, however if I scroll the UITableView down a page, the highlighted state becomes all buggy and shows rows highlighted that were once offscreen...... d*mn UITableViews....

Any thoughts?

03/24/2009, 01:09:02 – Like – Reply

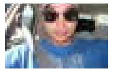
Michael Potter

Great tutorial. I always check your site for rock-solid Objective-C/Cocoa development insight. :)

I'm curious, however: how did you know to call setAnimationBeginsFromCurrentState:? What is the consequence of not calling this? Granted, I'm still a little noobish when it comes to CA...

06/25/2009, 15:21:35 – Like – Reply

Corey Floyd



Matt,

I've been using your heterogenous cells and this post as a template to create my table view controllers. It works great.

I have been trying perform animated deletions as well, but with less success. Using the table view deletion methods are quite tricky to get right if you attempt to manipulate sections and rows simultaneously. I seem to run into quite a few edge cases that cause exceptions.

My current solution can be found here:
http://stackoverflow.com/questions/1061071/uitableview-deleting-sections-with-animation

...which works most of the time, but I have been getting issues as of late.

Have you solve this in a generic method as well?
06/29/2009, 20:51:00 – Like – Reply

Eric Cooper

Matt,

Thanks for all your posts! I have one question about this one: is there any way to change the "blue selection color" to something arbitrary? I am trying to build an app for someone and it is very important to them that the selection color be a desaturated orange color.

Thanks.
10/06/2009, 19:43:11 – Like – Reply

Matt Gallagher

The blue selection color in this example is chosen on line 166 of MultiSelectC ellController.m.

But this example deliberately doesn't use the default selection approach on the iPhone. If you're actually asking about the default row selection color on the iPhone, then you can set the selectedBackgroundView on any cell to a view of your chosing and it will display that view when it is selected.
10/06/2009, 19:50:01 – Like – Reply

Eric Cooper

I had been trying to incorporate some of your code while using the undocumented UITableViewCellEditingStyle 3. While this approach got me fairly far fairly quickly, the setting of the selection color was just too difficult.

So, having adopted your abstracted cell controller design, I am wondering where/how one would implement "Delete".

If I make public your updateAndReload: method (by adding declaration to GenericTableViewController) and call it from my delete: method, then deletion works.

But I wonder if this is how it is intended to work.

Thanks again.
10/07/2009, 17:37:21 – Like – Reply

< Prev 1 2 3 Next >

Leave a comment

Social Networking by Echo

Basic HTML formatting tags (<a>, <b>, <i>, and <pre>) are permitted in comments.

Newer Post Older Post Home