Chapter 9
Editing UITableView

- Editing Mode
- Adding Rows
- Deleting Rows
- Moving Rows

Editing Mode

- UITableView has an editing property that when this property is set to YES, the UITableView enters editing mode
- Editing mode allows the user to add new rows, delete, and reorder existing ones
- Editing mode does not allow the user to edit the content of a row.

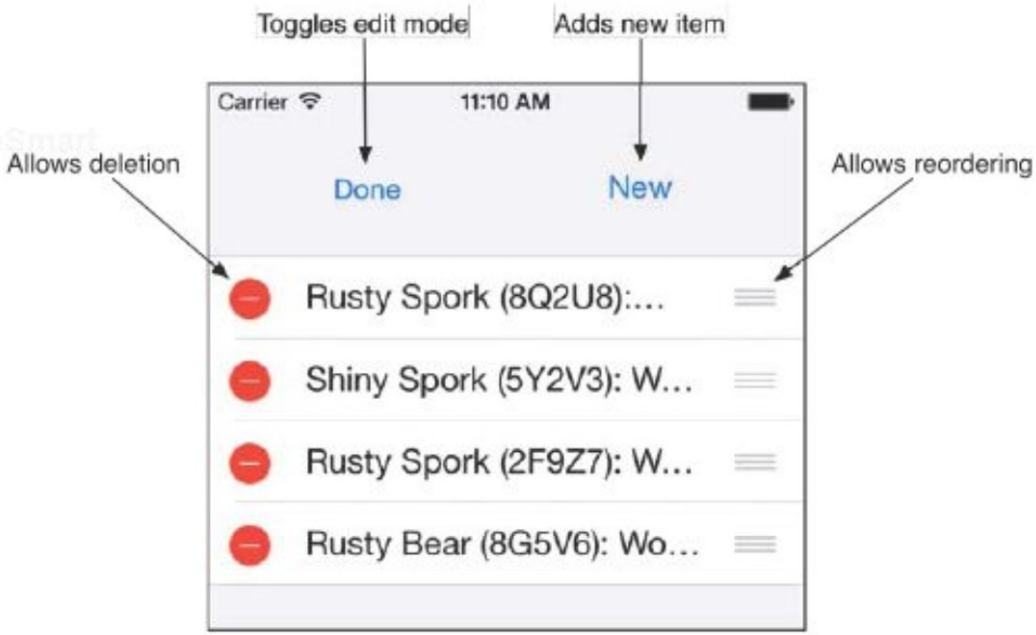


Table Header and Footer

- For every UITableView we can have a header and a footer view.
- Header and footer views can be any instance of UIView
- A header view appears at the top of a table and is useful for adding section-wide or table-wide titles and controls.
- There can be a table header/footer and a section header/footer
- We use the header view to add an “Edit” button that puts the table in edit mode

Revisiting Homeowner

- In (a copy of) Homeowner project I will modify INIItemsViewController so that

  it will have a property referencing the view that will be placed in the header

  it will have two actions that will be the targets for a button to toggle edit mode and one to add a new row

```
9  #import "INIItemsViewController.h"
10 #import "INIItem.h"
11 #import "INIItemStore.h"
12
13 @interface INIItemsViewController ()
14
15 @property (nonatomic,strong) IBOutlet UIView * headerView;
16
17 @end
18
19 @implementation INIItemsViewController ••• @end
78
```

```
67 - (IBAction) addNewItem:(id) sender
68 {
69     // Will add needed code here
70 }
71
72 - (IBAction) toggleEditingMode:(id) sender
73 {
74     // Will add needed code here
75 }
```

More on .XIB files

- We will create the new XIB file.
- Unlike the previous XIB files we created, this XIB file will not deal with the view controller's view.
- In addition to using XIB files to create the view for a view controller, we also use them any time we want to lay out view objects, archive them, and have them loaded at runtime.
- In this example we will use a .XIB file to create the view that will be layered over the header subview.

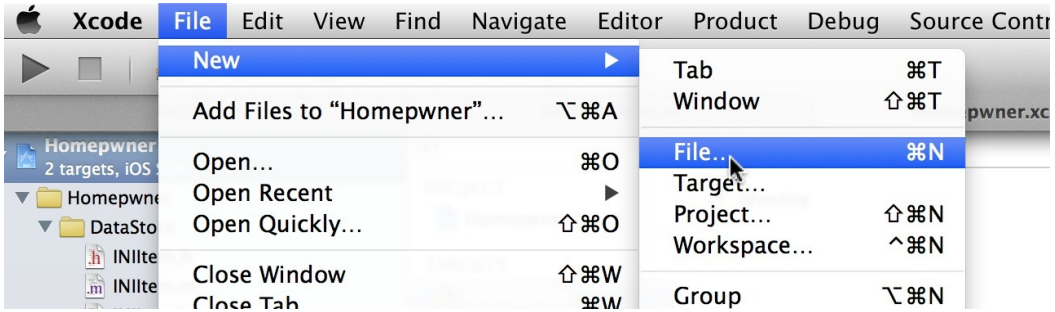
Creating the Header View

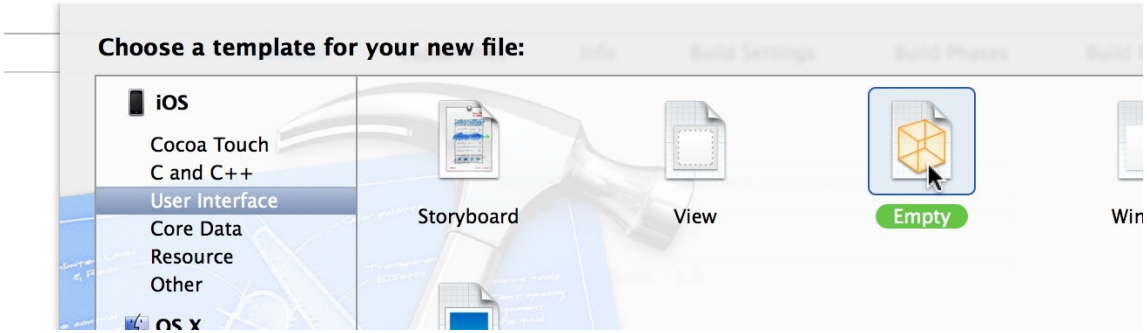
- The header view will be an instance of UIView named headerView.
- headerView will be archived in .xib file named headerView.xib
- headerView must have itemsViewController as its File's Owner
- headerView will have two buttons, an "Edit" and a "New" button.

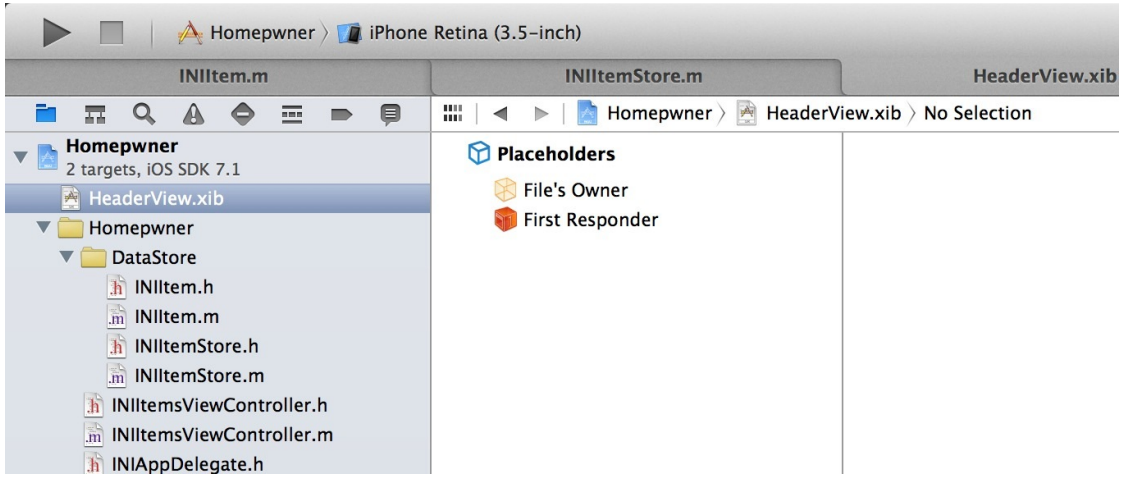
  The "Edit" button will have itemsViewController as a target to implement the method needed to toggle to editing mode.

  The "New" button will have itemsViewController as a target to implement the method needed to add a new item

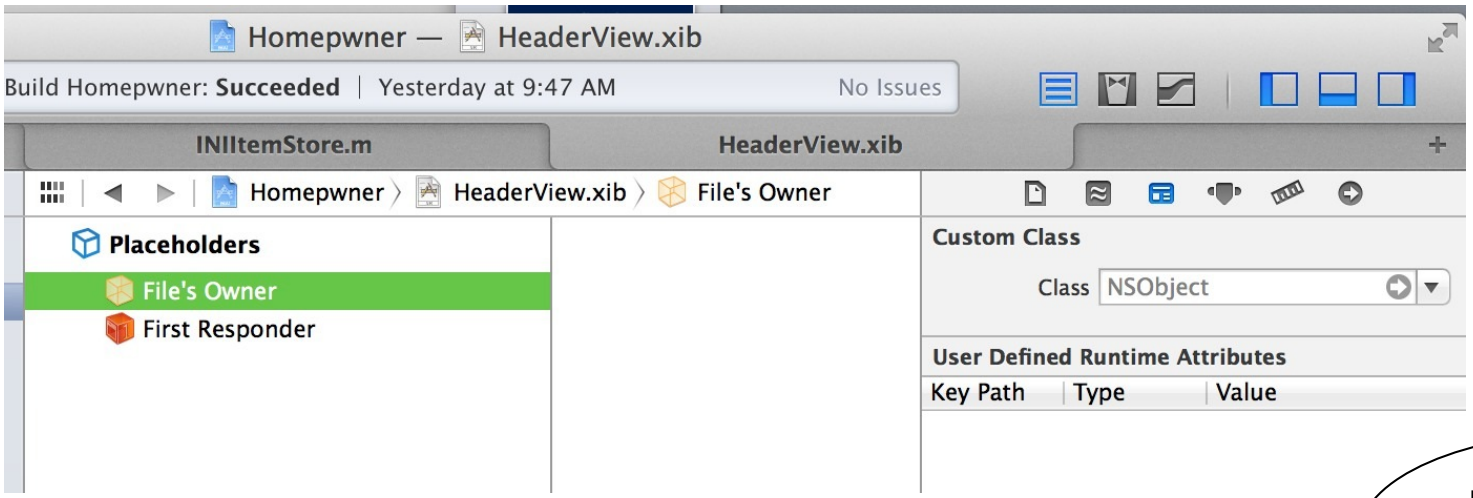
Creating the new .XIB



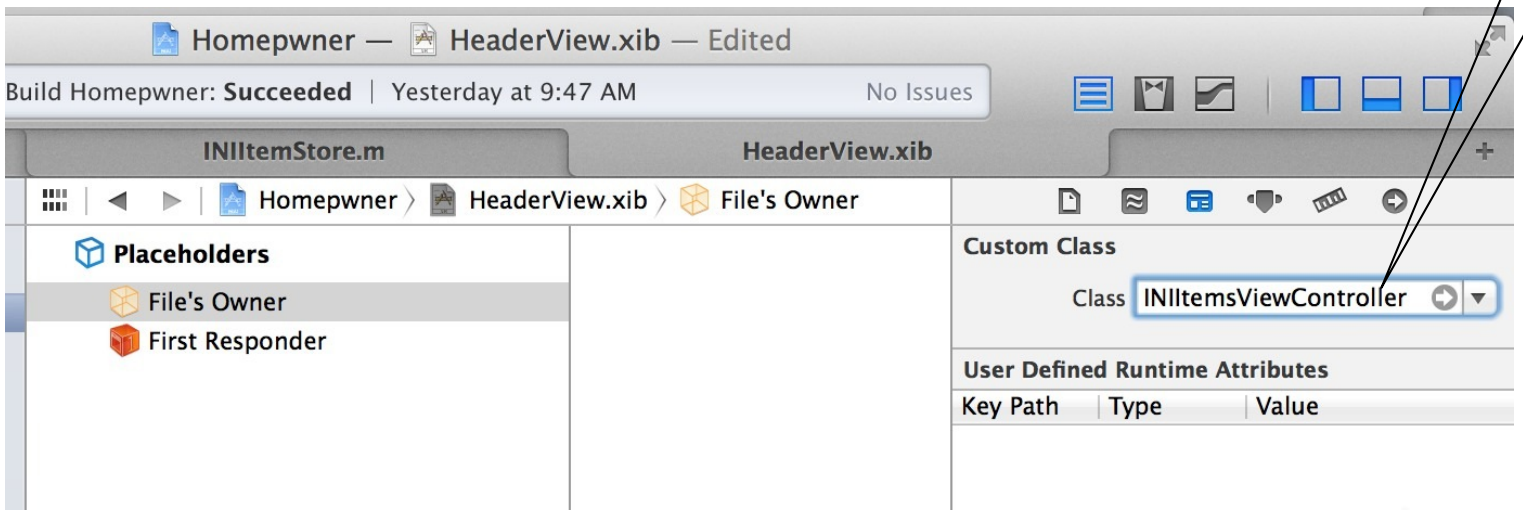




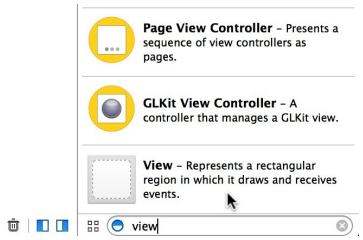
The Correct File's Owner

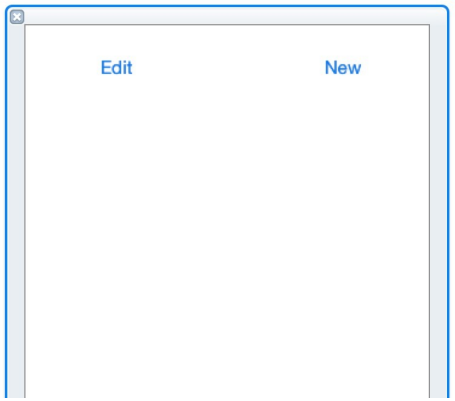


Dropping the File's Owner Class

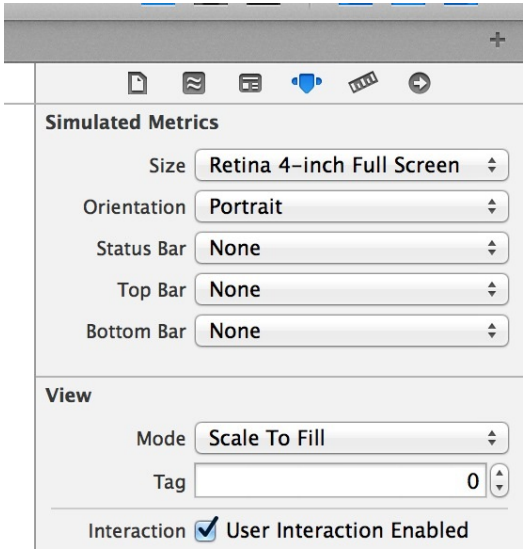


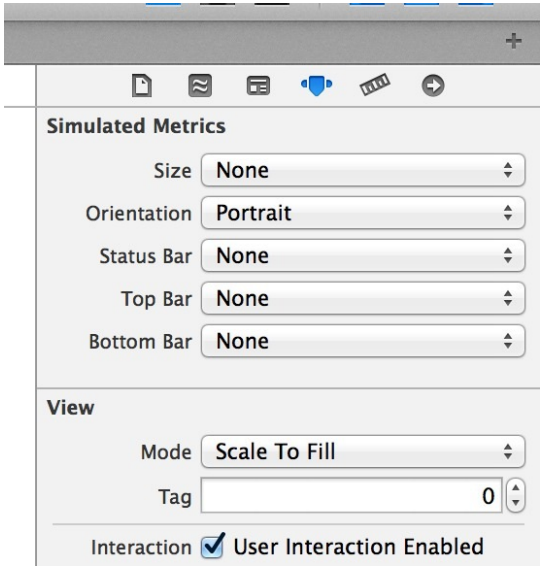
headerView.xib



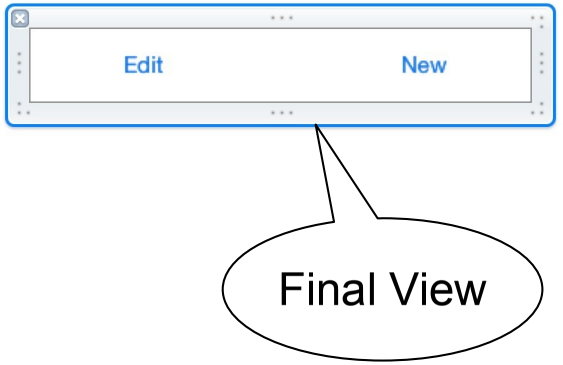


Original View Size is locked

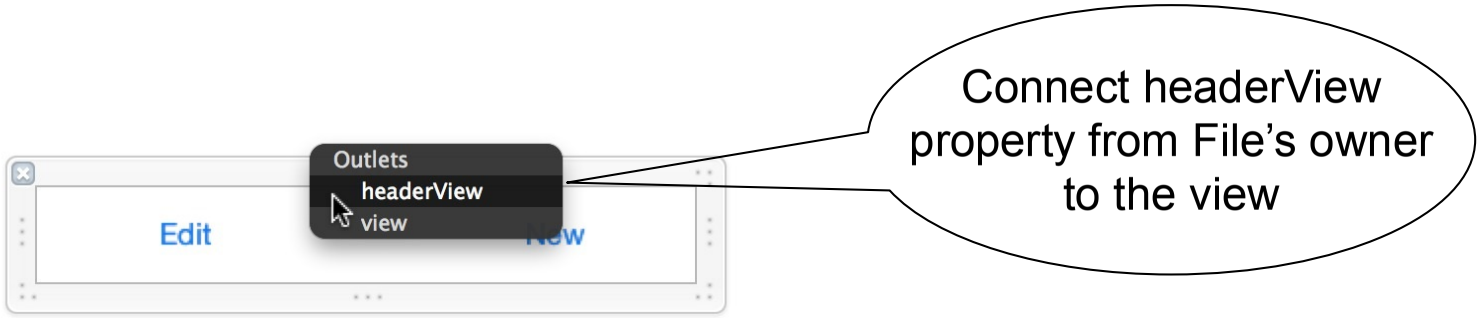




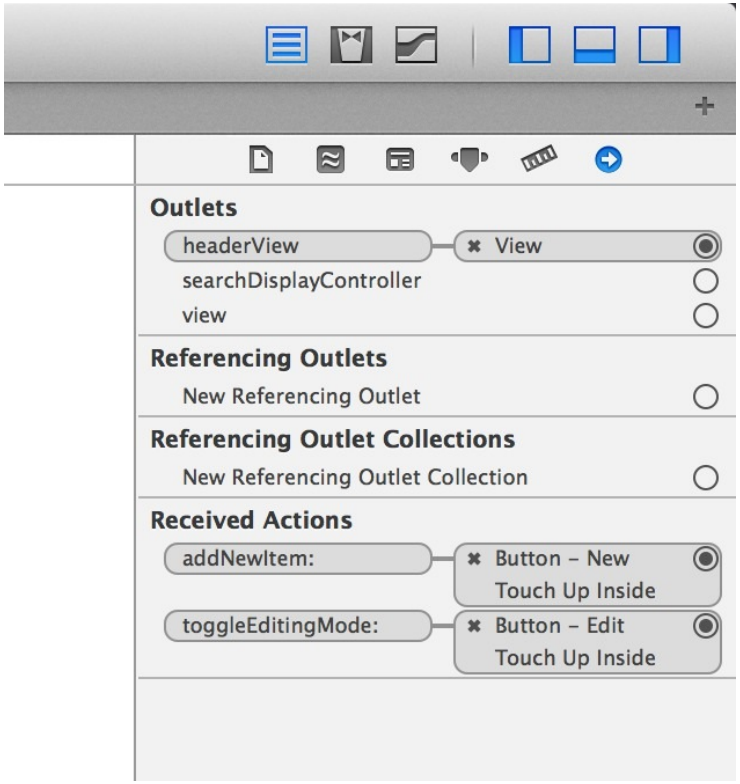
Unlock the size

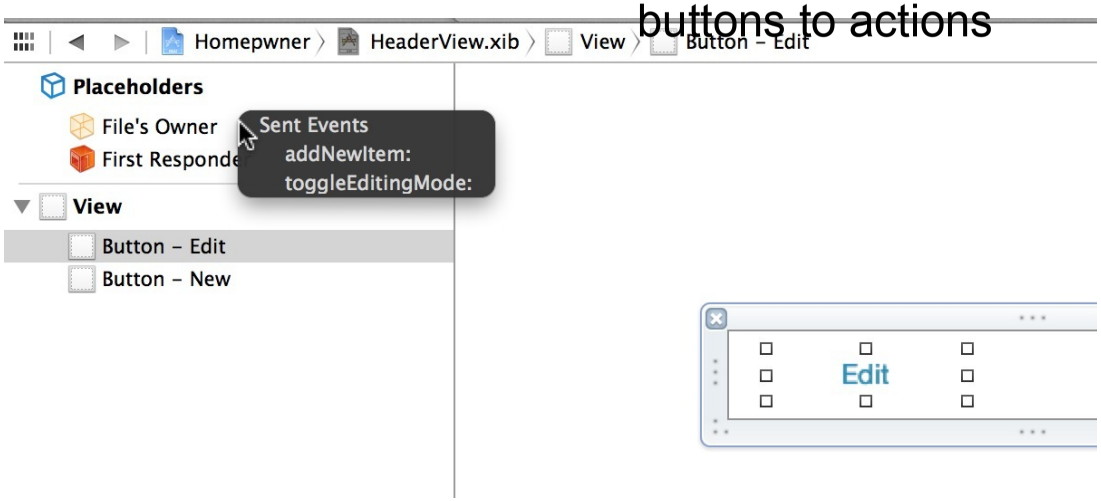


Making the connections



Connect buttons to actions



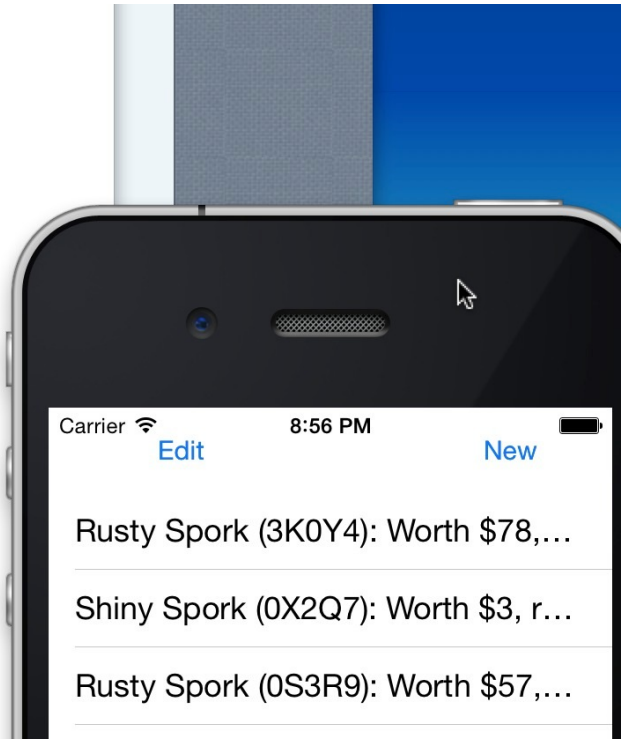


Final Connections

Updating INIItemsViewController

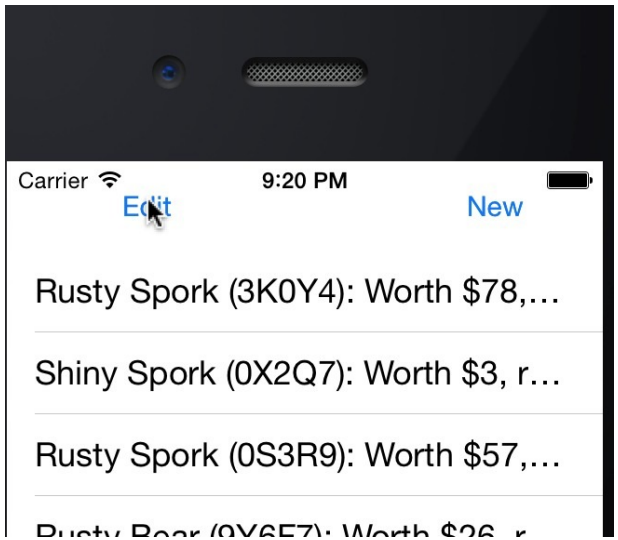
- So far we created the .XIB file which needs to be loaded by the application

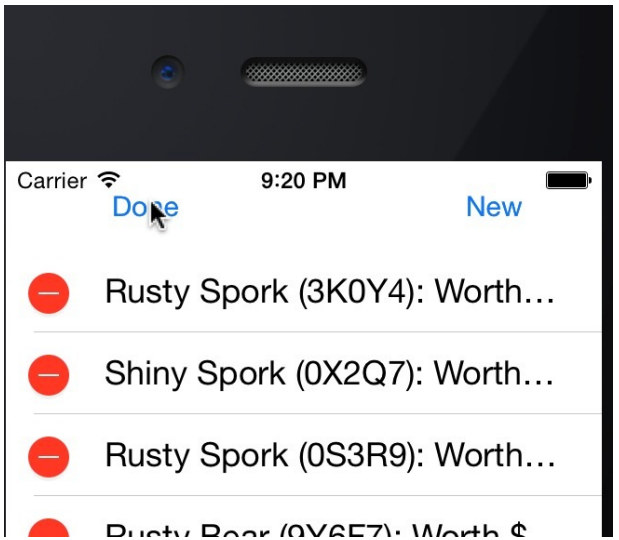
```
46  {...}
60
61  - (void) viewDidLoad
62  {
63      [super viewDidLoad];
64      [self.tableView registerClass:[UITableViewCell class]
            forCellReuseIdentifier:@"UITableViewCell"];
65
66      UIView *header = self.headerView;
67      [self.tableView setTableHeaderView: header];
68  }
69
70  - (IBAction) addNewItem:(id) sender
71  {
72      // Will add needed code here
73  }
74
75  - (IBAction) toggleEditingMode:(id) sender
76  {
77      // Will add needed code here
78  }
79
80  - (UIView *) headerView
81  {
82      // If you have not loaded the headerView yet...
83      if (!_headerView) {
84          // Load HeaderView.xib
85          [[ NSBundle mainBundle] loadNibNamed:@"HeaderView" owner: self options: nil];
86      }
87      return _headerView;
88  }
89
```



Toggle Editing Mode

```
74
75  - (IBAction) toggleEditingMode:(id) sender
76  {
77      // If you are currently in editing mode...
78      if (self.isEditing) {
79
80          // Change text of button to inform user of state
81          [sender setTitle: @"Edit" forState: UIControlStateNormal];
82
83          // Turn off editing mode
84          [self setEditing: NO animated: YES];
85      } else {
86
87          // Change text of button to inform user of state
88          [sender setTitle:@"Done" forState: UIControlStateNormal];
89          // Enter editing mode
90          [self setEditing: YES animated: YES];
91      }
92  }
93
```





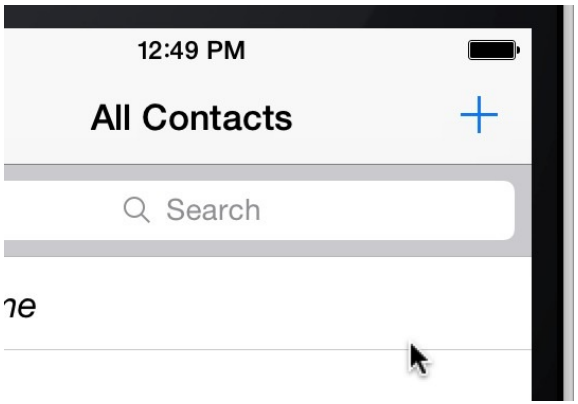
Adding Rows

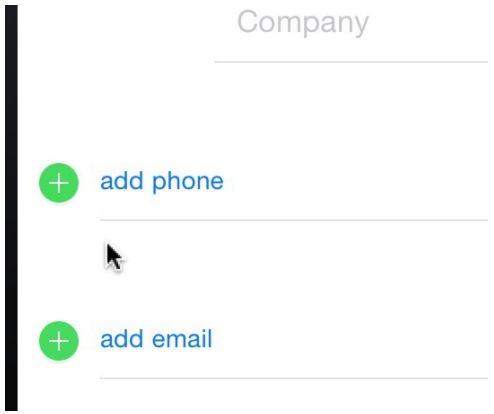
- There are two common interfaces for adding rows to a table view at runtime.

  A button above the cells of the table view. This is usually for adding a record for which there is a detail view. For example, in the Contacts app, you tap a button when you meet a new person and want to take down all their information.

  A cell with a green plus sign. This is usually for adding a new field to a record, such as when you want to add a birthday to a person's record in the Contacts app. In edit mode, you tap the green plus sign next to " add birthday".

- In this exercise, you are using the New button in the header view instead. When this button is tapped, a new row will be added to the UITableView.

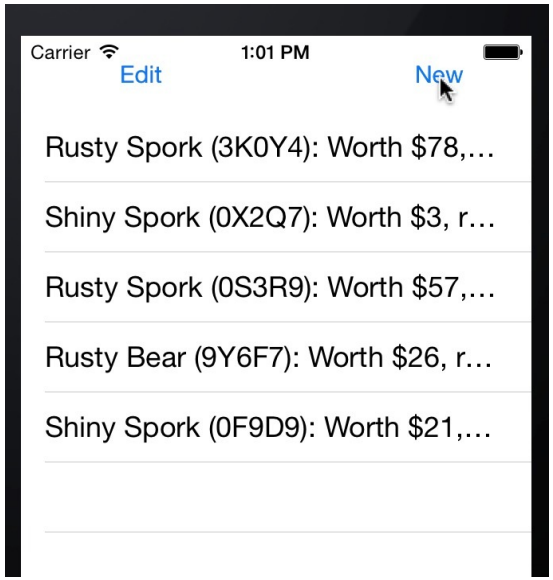


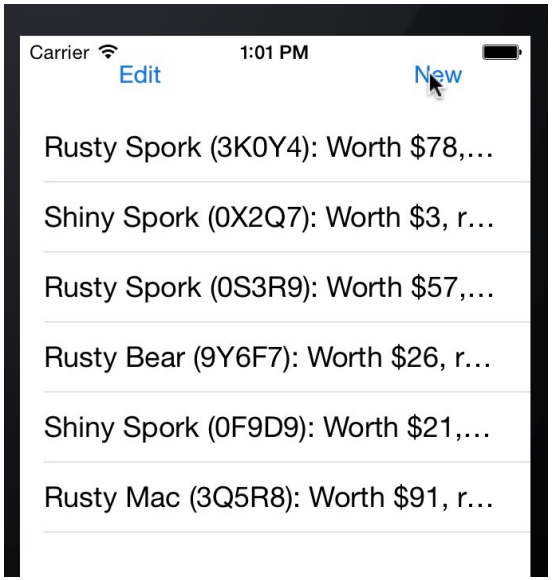


Updating itemsViewController, completing the target-action pattern

```
69
70  - (IBAction) addNewItem:(id) sender
71  {
72      // NSInteger lastRow = [[self tableView] numberOfRowsInSection: 0];  // This will crash the app
73      // Create a new BNRItem and add it to the store
74
75      INIItem *newItem = [[INIItemStore sharedStore] createItem];
76
77      // Figure out where that item is in the array
78      NSInteger lastRow = [[[INIItemStore sharedStore] allItems] indexOfObject: newItem];
79      NSIndexPath *indexPath = [NSIndexPath indexPathForRow: lastRow inSection: 0];
80
81      // Insert this new row into the table
82      [self. tableView insertRowsAtIndexPaths:@[indexPath] withRowAnimation: UITableViewRowAnimationTop];
83  }
```

You must make sure that UITableView and it's dataSource agree on the number of rows, so add the new item to the dataStore first then determine where it is in the array





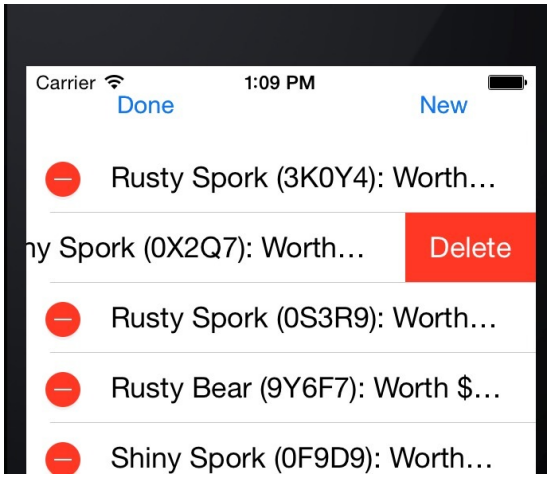
Deleting Rows

- Deleting a row is a two steps process:

  touch the little red circle with the - sign to indicate you desire to delete the corresponding row.

  A confirmation red delete button will show asking you to confirm your desire, and if you do the row will be deleted.

- The row to be deleted has to be removed from the dataStore and the table view.



Deleting Rows

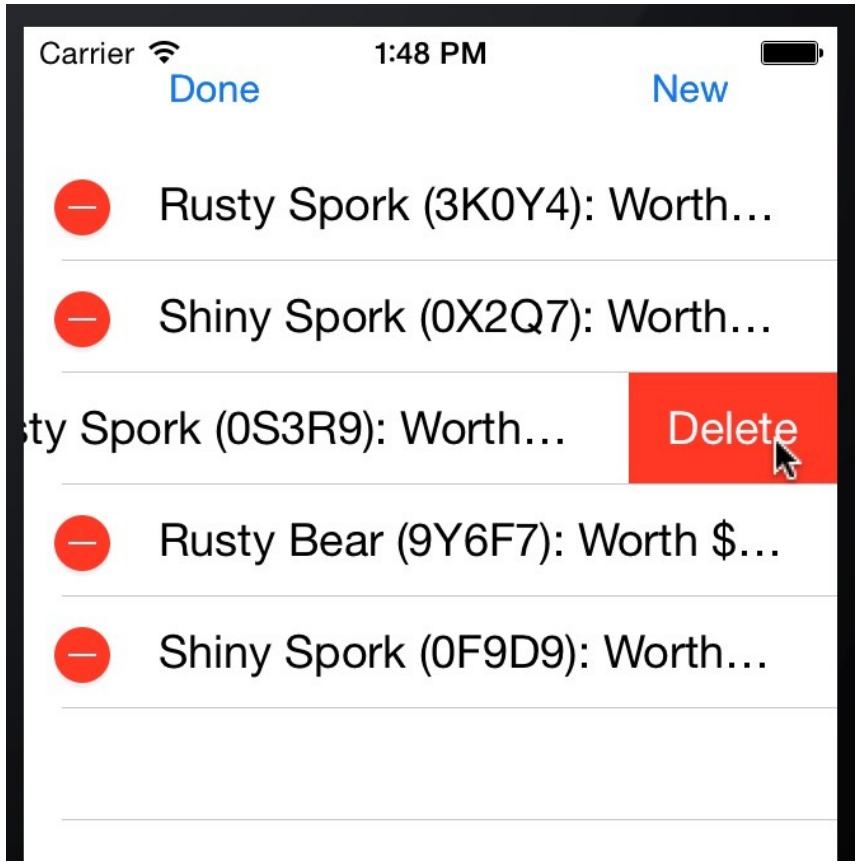
- In the INIItemStore add “removeItem:” method and remove the ***identical*** object to the one you are passing in the argument.
- Implement tableView:commitEditingStyle:forRowAtIndexPath: using deleteRowsAtIndexPaths:withRowAnimation:

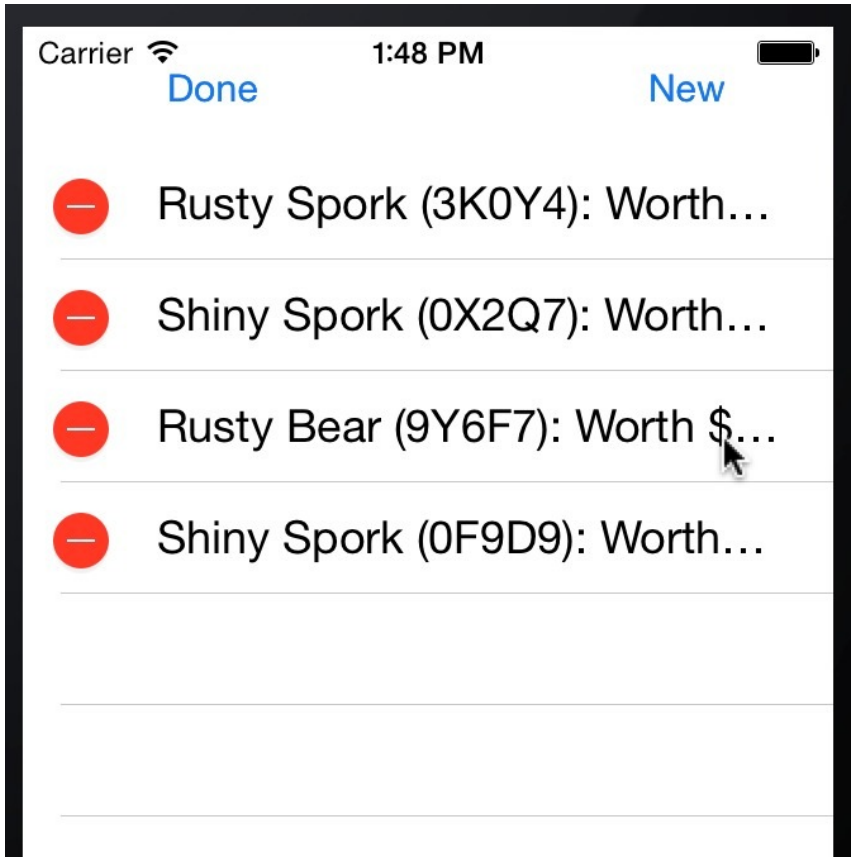
```
#import <Foundation/Foundation.h>
#import "INIItem.h"

@interface INIItemStore : NSObject

@ property (nonatomic, readonly, copy) NSArray * allItems;

// Notice that this is a class method and prefixed with a + instead of a -
+ (instancetype) sharedStore;
- (INIItem *)createItem;
- ( void)removeItem:(INIItem *) item;

@end
```

```
- (NSArray *) allItems
{ ... }

- (INIItem *) createItem
{ ... }

- (void) removeItem:(INIItem *) item
{
    [self.privateItems removeObjectIdenticalTo: item];
}

@end
```

```
- (void) tableView:(UITableView *)tableView commitEditingStyle:(UITableViewCellEditingStyle) editingStyle forRowAtIndexPath:(NSIndexPath *) indexPath
{
    // If the table view is asking to commit a delete command...
    if (editingStyle == UITableViewCellEditingStyleDelete) {

        NSArray *items = [[INIItemStore sharedStore] allItems];     //Get the items array
        INIItem *item = items[indexPath. row];                      //Get the item
        [[INIItemStore sharedStore] removeItem: item];              //Remove the (identical) item

        // Also remove that row from the table view with an animation
        [tableView deleteRowsAtIndexPaths:@[indexPath] withRowAnimation: UITableViewRowAnimationFade];
    }
}
```

Testing Delete Item





More on removing objects

- You could use NSMutableArray's removeObject: method instead of removeObjectIdenticalTo:, but consider the difference:

  removeObject: goes to each object in the array and sends it the message isEqual:.

  A class can implement this method to return YES or NO based on its own determination.

  For example, two INIItem objects could be considered equal if they had the same valueInDollars.
- The method removeObjectIdenticalTo: removes an object if and only if it is the exact same object as the one passed in this message.
- While INIItem does not currently override isEqual:, it could in the future. Therefore, you should use removeObjectIdenticalTo: when you are specifying a particular instance.

Moving Rows

- Since the table reflects the order in which items are stored in the dataStore moving an item requires:
  - Changing the items order in the dataStore, i.e. moving it from index i to index j
  - Changing the table look so that the item moves from current spot to the destination spot.

```
67  }
68
69  - (void) moveItemAtIndex:(NSUInteger) fromIndex toIndex:( NSUInteger) toIndex
70  {
71      if ( fromIndex == toIndex) {
72          return;
73      }
74      // Get pointer to object being moved so you can re-insert it
75      INIItem *item = self.privateItems[fromIndex];
76
77      // Remove item from array
78      [self.privateItems removeObjectAtIndex: fromIndex];
79
80      // Insert item in array at new location
81      [self.privateItems insertObject: item atIndex: toIndex];
82  }
83
```

```
16  // Notice that this is a class method and prefixed with a + i
17  + (instancetype) sharedStore;
18  - (INIItem *)createItem;
19  - ( void)removeItem:(INIItem *) item;
20  - (void) moveItemAtIndex: (NSUInteger)fromIndex toIndex: (NSU
21
```

removeObjectAtIndex:

Removes the object at *index* .

```
- (void)removeObjectAtIndex:(NSUInteger)index
```

Parameters

index

The index from which to remove the object in the array. The value must not exceed the bounds of the array.

Important: Raises an `NSRangeException` if *index* is beyond the end of the array.

Discussion

To fill the gap, all elements beyond *index* are moved by subtracting 1 from their index.

Availability

Available in iOS 2.0 and later.

See Also

Moving Rows

- Changing the table look so that the item moves from current spot to the destination spot.

```
- (void) tableView:(UITableView *)tableView moveRowAtIndexPath:(NSIndexPath *)sourceIndexPath toIndexPath:(NSIndexPath *)destinationIndexPath
{
    [[INIItemStore sharedStore] moveItemAtIndex: sourceIndexPath.row toIndex: destinationIndexPath.row];
}
```