ChopinLiszt 9/4/09

```
//  ChopinLisztAppDelegate.h -- Application Delegate object for ChopinLiszt application

//  Created by Charles Cranston on 11/7/08.
//  Copyright Charles Cranston 2008. All rights reserved.

#import <UIKit/UIKit.h>
#import "LisztController.h"

@interface ChopinLisztAppDelegate : NSObject <UIApplicationDelegate> {
    IBOutlet UIWindow *window;
    IBOutlet UITabBar *tabBar;
    IBOutlet LisztController *rootViewController;
}

@property (nonatomic, retain) UIWindow *window;
@property (nonatomic, retain) UITabBar *tabBar;
@property (nonatomic, retain) LisztController *rootViewController;

@end
```

```
//  ChopinLisztAppDelegate.m -- Application Delegate object for ChopinLiszt application

//  Created by Charles Cranston on 11/7/08.
//  Copyright Charles Cranston 2008. All rights reserved.

#import "ChopinLisztAppDelegate.h"

@implementation ChopinLisztAppDelegate

@synthesize window;
@synthesize tabBar;
@synthesize rootViewController;

-(void) dealloc {
    [rootViewController release];
    [tabBar release];
    [window release];
    [super dealloc];
}

// Add tab bar and (view of) view controller to window and activate

-(void) applicationDidFinishLaunching: (UIApplication *) application {
    [window addSubview:[rootViewController view]];
    tabBar.selectedItem = [tabBar.items objectAtIndex:0];
    [window addSubview:tabBar];
    [window makeKeyAndVisible];
}

// Cause application data to be rewritten to data file

-(void) applicationWillTerminate: (UIApplication *) application {
    [rootViewController saveData];
}

@end
```

```
//  LisztController.h -- Custom subclass of UITableViewController for ChopinLiszt application

//  Created by Charles Cranston on 11/16/08.
//  Copyright 2008 Charles Cranston. All rights reserved.

#import <UIKit/UIKit.h>

@interface LisztController: UITableViewController <UITextFieldDelegate,UITabBarDelegate> {
    NSMutableArray *itemLiszt;          // Main data base for view controller
    NSMutableArray *showLiszt;          // Either another pointer to itemLiszt object or a subset copy
}

-(void) saveData;

@end
```

```
//  LisztController.m -- Custom subclass of UITableViewController for ChopinLiszt application

//  Created by Charles Cranston on 11/16/08.
//  Copyright 2008 Charles Cranston. All rights reserved.

#import "LisztController.h"
#import "LisztItem.h"

#define LAZYLOAD(item,source) if (nil==item) {source;}

// Name of file we store our data into

#define kFilename @"ChopinLisztData"

// Position of add button in list cell

// #define kButtLeft  220
#define kButtLeft  230

// Parameters for UITextField in list cell (to make it line up nicely with standard cell text)

#define kTextFieldLeft       45
#define kTextFieldFontSize   20
// NB: EditField.xib UITextField is 280 wide, not 314, so clear button not outside of content view.

// Amount to shorten the list to make room for keyboard while editing

#define kOFFSET_FOR_KEYBOARD 220

// These MUST correspond with tab bar item tags in MainWindow.xib

enum {kShopping=1, kStockTaking=2, kEditing=3};

// Private instance variables

UIImage *neededImage   = nil;               // Unchecked image (loaded at startup)
UIImage *unneededImage = nil;               // Checked (lazy load first enter stock taking mode)
UIImage *addImage      = nil;               // Image for add new entry button (lazy load enter edit mode)
UIImage *blobImage     = nil;               // Gray blob for inline editing (lazoload inline editing)
LisztItem *editItem    = nil;               // LisztItem currently under inline editing or nil
UITextField *editField = nil;               // UITextField lazy created for first inline edit


@implementation LisztController

-(void) dealloc {
    if (showLiszt != itemLiszt) [showLiszt release];
    [itemLiszt release];
    [super dealloc];
}
```

```
// pragma page Inline Editing code

// Initiate inline editing on item at specified index
// The UITextField we build here is stuffed in by the cell supply routine (below)

-(void) beginInlineEditingAt: (NSIndexPath *) indexPath {
    editItem = (LisztItem *) [itemLiszt objectAtIndex:indexPath.row];       // Set global item under edit
    LAZYLOAD(editField,
      NSArray *nibStuff = [[NSBundle mainBundle] loadNibNamed:@"EditField" owner:self options:nil];
      editField = [[nibStuff objectAtIndex:0] retain];
      editField.clearButtonMode = UITextFieldViewModeWhileEditing;
      editField.font = [UIFont boldSystemFontOfSize:kTextFieldFontSize];
      CGRect frame = editField.frame;
      frame.origin.x = kTextFieldLeft;
      frame.origin.y = (self.tableView.rowHeight-frame.size.height+1)/2;
      editField.frame = frame;
    )
    LAZYLOAD(blobImage,blobImage=[UIImage imageNamed:@"grayblob.png"])
    editField.text = editItem.itemName;                                 // Set initial name for edit
    [self.tableView reloadData];                                        // Get new table cells for inline editing
    [UIView beginAnimations:nil context:NULL];                          // Begin animation
    [UIView setAnimationDuration:0.3];                                  // Set duration of animation
    CGRect rect = self.view.frame;                                      // Get frame of view (tableView)
    rect.size.height -= kOFFSET_FOR_KEYBOARD;                           // Shorten view by keyboard height
    self.view.frame = rect;                                             // Set shortened frame for view
    [self.tableView
      scrollToRowAtIndexPath:indexPath
      atScrollPosition: UITableViewScrollPositionMiddle
      animated:NO
    ];                                                                  // Scroll current item to "middle" of table
    [self.tableView setScrollEnabled:NO];                               // Disable scroll for inline editing duration
    [editField becomeFirstResponder];                                   // Tell it to handle keystrokes
    [UIView commitAnimations];                                          // Actually do the animation
}


#pragma mark -
#pragma mark <UITextFieldDelegate> Methods

// Cause textField to return when the "return" key is typed

-(BOOL) textFieldShouldReturn: (UITextField *) theTextField {
    [theTextField resignFirstResponder];                                // Cause UITextField to End Editing
    return YES;
}


// Transition out of inline editing mode and back to normal edit mode

-(void) textFieldDidEndEditing: (UITextField *) theTextField {
    editItem.itemName = [theTextField.text retain];                     // Copy new name to LisztItem
    editItem = nil;                                                     // Set global: no longer inline editing
    [self.tableView setScrollEnabled:YES];                              // Re-enable scrolling for view
    [UIView beginAnimations:nil context:NULL];                          // Begin animation
    [UIView setAnimationDuration:0.3];                                  // Set duration of animation
    [theTextField removeFromSuperview];                                 // Remove the UITextField from the table cell
    CGRect rect = self.view.frame;                                      // Get frame of view (tableView)
    rect.size.height += kOFFSET_FOR_KEYBOARD;                           // Lengthen the view by keyboard height
    self.view.frame = rect;                                             // Set lengthened frame for view
    [self.tableView reloadData];                                        // Get new table cells for non-inline editing
    [UIView commitAnimations];                                          // Actually do the animation
}


// Insert new item after specified index (transitions to inline editing on new item)

-(void) insertNewItemAt: (NSInteger) index {
    [itemLiszt insertObject:[LisztItem newItemNamed:@"?" needed:YES] atIndex:index];          // Insert new object
    [self.tableView reloadData];                                                  // Get tableView size updated!!
    [self beginInlineEditingAt:[NSIndexPath indexPathForRow:index inSection:0]];    // Start inline edit new item
}


// Handler for "add new item" button in edit mode cells
// Button tag is set to the insert point by the cell supply routine (below)

-(void) addAction: (id) sender {
    NSInteger index = ((UIButton *) sender).tag;        // Get insertion point from button tag
    [self insertNewItemAt:index];                       // Insert new LisztItem into itemLiszt
}
```

```
// pragma page UITableView delegate and source methods

#pragma mark -
#pragma mark Table Delegate and Data Source protocol methods

// There is only one section
// There are as many rows as there are entries in the show list

-(NSInteger)
  tableView:             (UITableView *) tableView
  numberOfRowsInSection: (NSInteger)     section {
    return [showLiszt count];
}


// Cell supply routine - helper routine to allocate normal cells

static NSString *PlainCellID  = @"PlainCell";
static NSString *ButtCellID = @"ButtonCell";

-(UITableViewCell *) getPlainCellFor: (UITableView *) tableView {
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:PlainCellID];
    if (nil == cell)
        cell = [[[UITableViewCell alloc] initWithFrame:CGRectZero reuseIdentifier:PlainCellID] autorelease];
    return cell;
}

// Cell supply routine - supply different cells depending on current operation modes
// For Shopping and Stock Taking modes, supply cell with checkmark image
// For Inline Editing supply plain cells for context but attach UITextField to accomplish editing
// For Editing (but not Inline Editing) supply cell with added button
//  tag of button (which is, after all, a view) holds insert point in case button is pressed
//  to recover button after cell reuse, button tag copied to cell tag (dummy value used at cell creation)

-(UITableViewCell *)
  tableView:             (UITableView *) tableView
  cellForRowAtIndexPath: (NSIndexPath *) indexPath {
    LisztItem *item = (LisztItem *)[showLiszt objectAtIndex:indexPath.row]; // Get LisztItem
    UITableViewCell *cell;
    if ( !tableView.editing ) {                                          // Shopping or StockTaking
        cell = [self getPlainCellFor:tableView];                             // Plain cell
        cell.image = item.needIt ? neededImage : unneededImage;              // Image depending if needed
        cell.text = item.itemName;                                           // Item name in cell
    } else if (nil != editItem) {                                        // Editing AND inline editing
        cell = [self getPlainCellFor:tableView];                             // Plain Cell
//        cell.image = nil;                                                    // No checkmarks while editing
        cell.image = blobImage;
        if (item != editItem) {                                              // IF NOT item we are actually editing
            cell.text = item.itemName;                                           // Item name in cell
        } else {                                                             // Item we are actually editing
            cell.text = nil;                                                     // UITextField instead of name
            [cell.contentView addSubview:editField];                             // Add UITextField as subview
        }
    } else {                                                             // Editing but NOT inline editing
        UIButton *addButt;
        if ( nil == (cell=[tableView dequeueReusableCellWithIdentifier:ButtCellID]) ) { // Create cell and button
            cell = [[[UITableViewCell alloc] initWithFrame:CGRectZero reuseIdentifier:ButtCellID] autorelease];
            addButt = [[UIButton alloc] initWithFrame:CGRectMake(kButtLeft,
              (tableView.rowHeight-addImage.size.height)/2,addImage.size.width,addImage.size.height)];
            [addButt setBackgroundImage:addImage forState:UIControlStateNormal];
            [addButt addTarget:self action:@selector(addAction:) forControlEvents:UIControlEventTouchUpInside];
            [cell.contentView addSubview:addButt];                           // Add button as subview of cell
            cell.tag = addButt.tag = 9999;                                   // Link cell and button with value
        }
        addButt = (UIButton *)[cell.contentView viewWithTag:cell.tag];   // Recover button as subview of cell
        cell.image = nil;                                                    // No checkmarks while editing
        cell.text = item.itemName;                                           // Item name in cell
        cell.tag = addButt.tag = indexPath.row + 1;                          // Link cell and button with index
        [cell.contentView bringSubviewToFront:addButt];                  // Show button IN FRONT OF text name
    }
    return cell;
}


// Reorder items within the list

-(void)
  tableView:          (UITableView *) tableView
  moveRowAtIndexPath: (NSIndexPath *) fromIndexPath
  toIndexPath:        (NSIndexPath *) toIndexPath {
    LisztItem *transportee = [[itemLiszt objectAtIndex:fromIndexPath.row] retain];
    [itemLiszt removeObjectAtIndex:fromIndexPath.row];
    [itemLiszt insertObject:transportee atIndex:toIndexPath.row];
}
```

```
// pragma page UITableView delegate and source methods (continued)

// Select item from list depends on current state
//  Editing      - selection starts up inline editing on item
//  Stock Taking - selection causes checkbox to be flipped
//  Shopping     - selection causes item to be removed from separate display array

-(void)
  tableView:               (UITableView *) tableView
  didSelectRowAtIndexPath: (NSIndexPath *) indexPath {
    [tableView deselectRowAtIndexPath:indexPath animated:NO];                              // Undo base selection
    if ( tableView.editing ) {                                                             // Editing
        if (nil != editItem) [editField resignFirstResponder];                               // Already inline, stop
        [self beginInlineEditingAt: indexPath];                                              // Begin inline edit
    } else if (showLiszt == itemLiszt) {                                                   // Stock Taking
        LisztItem *item = (LisztItem *)[itemLiszt objectAtIndex:indexPath.row];              // Get LisztItem
        item.needIt = !item.needIt;                                                          // Flip needed
        UITableViewCell *cell = [tableView cellForRowAtIndexPath:indexPath];                 // Get cell
        cell.image = [UIImage imageNamed: item.needIt ? @"unchecked.png" : @"checked.png"]; // Change image
        [[tableView cellForRowAtIndexPath:indexPath] setNeedsDisplay];                       // Set for redisplay
    } else {                                                                               // Shopping
        LisztItem *item = (LisztItem *)[showLiszt objectAtIndex:indexPath.row];              // Get LisztItem
        item.needIt = NO;                                                                    // No longer need it
        [showLiszt removeObjectAtIndex:indexPath.row];                                       // Remove from show list
        [tableView deleteRowsAtIndexPaths:[NSArray arrayWithObject:indexPath] withRowAnimation:YES];
    }
}


// Show delete button only if editing but not inline editing

-(BOOL)
  tableView:             (UITableView *) tableView
  canEditRowAtIndexPath: (NSIndexPath *) indexPath {
    return (nil == editItem);
}


// Handle item delete

-(void)
  tableView:          (UITableView *)               tableView
  commitEditingStyle: (UITableViewCellEditingStyle) editingStyle
  forRowAtIndexPath:  (NSIndexPath *)               indexPath {
    if (editingStyle == UITableViewCellEditingStyleDelete) {
        [itemLiszt removeObjectAtIndex:indexPath.row];
        [tableView deleteRowsAtIndexPaths:[NSArray arrayWithObject:indexPath] withRowAnimation:YES];
    }
    if (editingStyle == UITableViewCellEditingStyleInsert) { NSLog (@"Inconsistency!"); }
}


#pragma mark -
#pragma mark Tab Bar Delegate methods

// This is called when a tab bar item is pressed, to change operating mode
// For shopping mode showLiszt is separate array, else another reference to itemLiszt array
// Recovery from deleting everything: in edit mode if no items add one

-(void)
tabBar:        (UITabBar *)     tabBar
didSelectItem: (UITabBarItem *) item {
    if ( showLiszt != itemLiszt) [showLiszt release];              // IFF was displaying subset, release array
    [UIView beginAnimations:nil context:NULL];
    [UIView setAnimationDuration:0.3];
    [UIView setAnimationTransition:UIViewAnimationTransitionFlipFromLeft forView:self.tableView cache: NO];
    switch (item.tag) {
      case kShopping:
        showLiszt = [[NSMutableArray alloc] initWithObjects: nil];                 // Make list to show
        for (LisztItem *item in itemLiszt) if (item.needIt) [showLiszt addObject:item]; // Add needed items
        [self.tableView setEditing:NO animated:NO];                                // Table not editing
        break;
      case kStockTaking:
        LAZYLOAD(unneededImage,unneededImage=[UIImage imageNamed:@"checked.png"])  // Ensure image loaded
        showLiszt = itemLiszt;                                                     // Show directly from itemLiszt
        [self.tableView setEditing:NO animated:NO];                                // Table not editing
        break;
      case kEditing:
        LAZYLOAD(addImage,addImage=[UIImage imageNamed:@"addbutton.png"])          // Ensure image loaded
        showLiszt = itemLiszt;                                                     // Show directly from itemLiszt
        [self.tableView setEditing:YES animated:NO];                               // Table is editing
        if ( 0 == [itemLiszt count] ) [self insertNewItemAt:0];                    // If empty, create one item
    }
    [self.tableView reloadData];                                   // Reload table data for dispay
    [UIView commitAnimations];                                     // Start the animation
}
```

```
// pragma page Loading and saving of application data

-(NSString *) dataFilePath {
    return [
      [NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,NSUserDomainMask,YES) objectAtIndex:0]
      stringByAppendingPathComponent:kFilename
    ];
}


// Load LisztItems from saved file, or supply initial configuration, setup for shopping mode

-(void) viewDidLoad {
    NSString *filePath = [self dataFilePath];
    if ([[NSFileManager defaultManager] fileExistsAtPath: filePath]) {
        NSData *data = [[NSData alloc] initWithContentsOfFile:filePath];
        NSKeyedUnarchiver *reader = [[NSKeyedUnarchiver alloc] initForReadingWithData:data];
        itemLiszt = [[reader decodeObjectForKey:@"ItemLiszt"] retain];
        [reader finishDecoding];
        [reader release];
        [data release];
    } else {
        itemLiszt = [[NSMutableArray alloc] initWithObjects:        // temp for debugging
          [LisztItem newItemNamed:@"Money"              needed:NO ],
          [LisztItem newItemNamed:@"Lunch Meat 1"       needed:NO ],
          [LisztItem newItemNamed:@"Margarine 1"        needed:NO ],
          [LisztItem newItemNamed:@"Yogurt 1"           needed:NO ],
          [LisztItem newItemNamed:@"Orange Juice 1"     needed:NO ],
          [LisztItem newItemNamed:@"Eggs 1"             needed:NO ],
          [LisztItem newItemNamed:@"Milk 1"             needed:YES],
          [LisztItem newItemNamed:@"Cheese 1"           needed:NO ],
          [LisztItem newItemNamed:@"Peanut Butter 2"    needed:NO ],
          [LisztItem newItemNamed:@"Saltines 3"         needed:NO ],
          [LisztItem newItemNamed:@"Tuna 4"             needed:NO ],
          [LisztItem newItemNamed:@"Mayonnaise 4"       needed:NO ],
          [LisztItem newItemNamed:@"Cereal 5"           needed:NO ],
          [LisztItem newItemNamed:@"Popcorn 5"          needed:NO ],
          [LisztItem newItemNamed:@"Oatmeal 5"          needed:NO ],
          [LisztItem newItemNamed:@"Rye Bread 8"        needed:YES],
          [LisztItem newItemNamed:@"English Muffins 8"  needed:NO ],
          [LisztItem newItemNamed:@"Toilet Paper 10"    needed:NO ],
          [LisztItem newItemNamed:@"Soda 19"            needed:NO ],
          [LisztItem newItemNamed:@"Desserts 23"        needed:NO ],
          [LisztItem newItemNamed:@"Dinners 24"         needed:NO ],
          [LisztItem newItemNamed:@"Batteries"          needed:NO ],
        nil];
    }
    showLiszt = [[NSMutableArray alloc] initWithObjects: nil];                      // Make list for showing
    for (LisztItem *item in itemLiszt) if (item.needIt) [showLiszt addObject:item]; // Add needed items
    neededImage = [UIImage imageNamed:@"unchecked.png"];                            // Load needed image
}


// Save data to file before exiting

-(void) saveData {
    NSMutableData *data = [[NSMutableData alloc] init];                     // Allocate data block
    NSKeyedArchiver *writer = [[NSKeyedArchiver alloc] initForWritingWithMutableData:data];
    [writer encodeObject:itemLiszt forKey:@"ItemLiszt"];                    // Save the data
    [writer finishEncoding];                                                // End the save process
    [data writeToFile:[self dataFilePath] atomically:YES];                  // Write archive to file
    [writer release];                                                       // Clean up archiver
    [data release];                                                         // Clean up data block
}

@end
```

```
//  LisztItem.h -- Custom object for storing shopping item status for ChopinLiszt application

//  Created by Charles Cranston on 12/14/08.
//  Copyright Charles Cranston 2008. All rights reserved.

#import <UIKit/UIKit.h>

@interface LisztItem : NSObject {
    NSString* itemName;
    BOOL      needIt;
}

@property (nonatomic, retain) NSString* itemName;
@property (nonatomic, assign) BOOL      needIt;

+(LisztItem *)
  newItemNamed: (NSString *) itemName
  needed:       (BOOL)       needed;

-(id) initWithCoder: (NSCoder *) decoder;
-(void) encodeWithCoder: (NSCoder *) encoder;

@end
```

```
//  LisztItem.m -- Custom object for storing shopping item status for ChopinLiszt application

//  Created by Charles Cranston on 12/14/08.
//  Copyright 2008 Charles Cranston. All rights reserved.

#import "LisztItem.h"

@implementation LisztItem

@synthesize itemName;
@synthesize needIt;

-(void) dealloc {
    [itemName release];
    [super dealloc];
}

#define kItemName @"ItemName"
#define kNeedIt   @"NeedIt"

-(id) initWithCoder: (NSCoder *) decoder{
    if ( self=[super init] ) {
        self.itemName = [decoder decodeObjectForKey:kItemName];
        needIt        = [decoder decodeBoolForKey:  kNeedIt  ];
    }
    return self;
}

-(void) encodeWithCoder: (NSCoder *) encoder{
    [encoder encodeObject:itemName forKey:kItemName];
    [encoder encodeBool:  needIt   forKey:kNeedIt  ];
}

+(LisztItem *)
  newItemNamed: (NSString *) newItemName
  needed:       (BOOL)       needed {
    LisztItem *item = [[LisztItem alloc] init];
    if (item) {
        item.itemName = newItemName;
        item.needIt = needed;
    }
    return item;
}

@end
```