

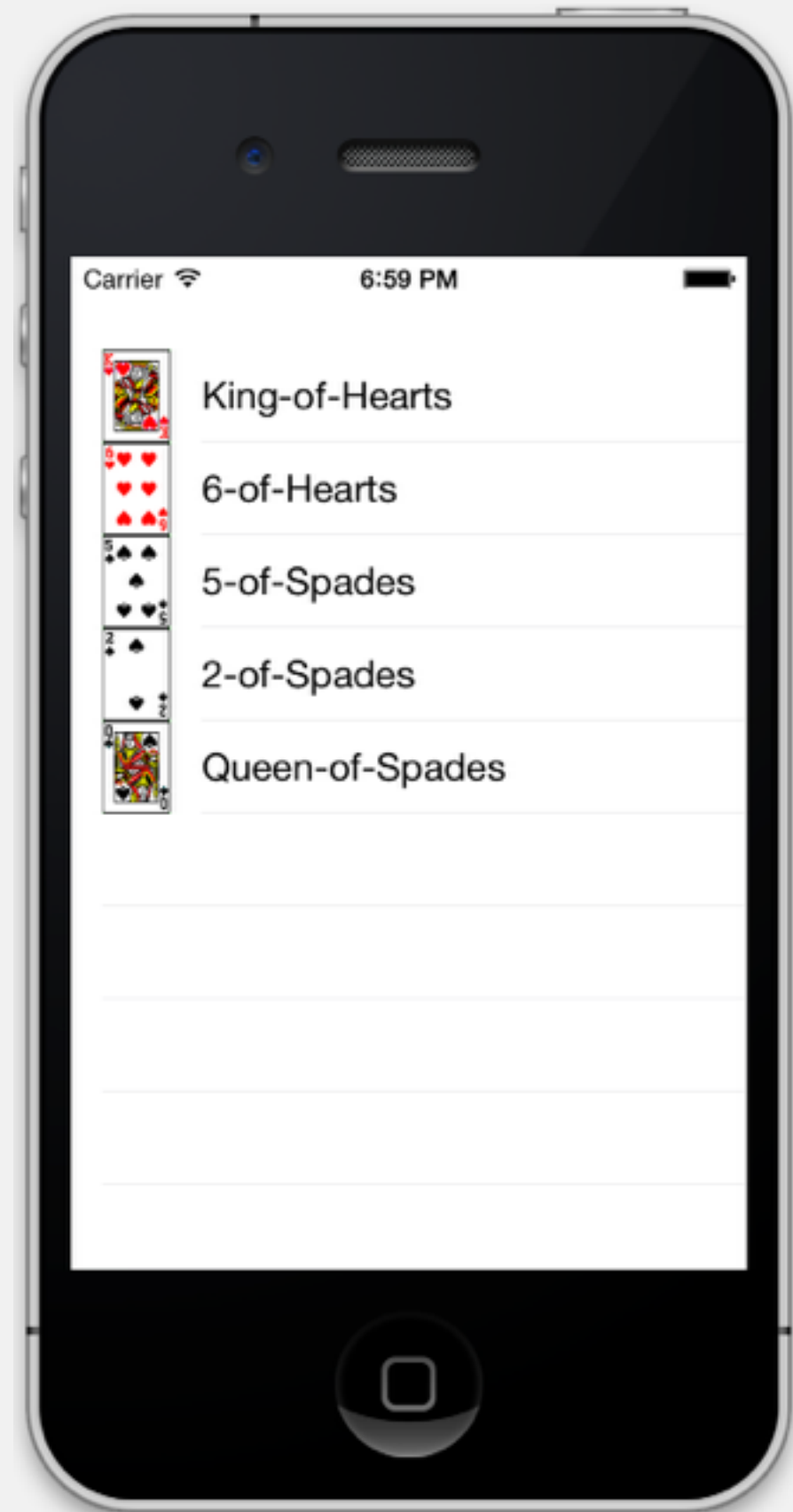
Table View Pain Points

Ken Auer

ken.auer@rolemodelsoftware.com
[@kauerrolemodel](https://twitter.com/kauerrolemodel)

Section

The Missing Object



```

@implementation RMSViewController

static NSString *cardCellIdentifier = @"CardCell";

-(RMSGoFishGame *)game {
    if (!_game) {
        _game = [RMSGoFishGame new];
        [_game deal];
    }
    return _game;
}

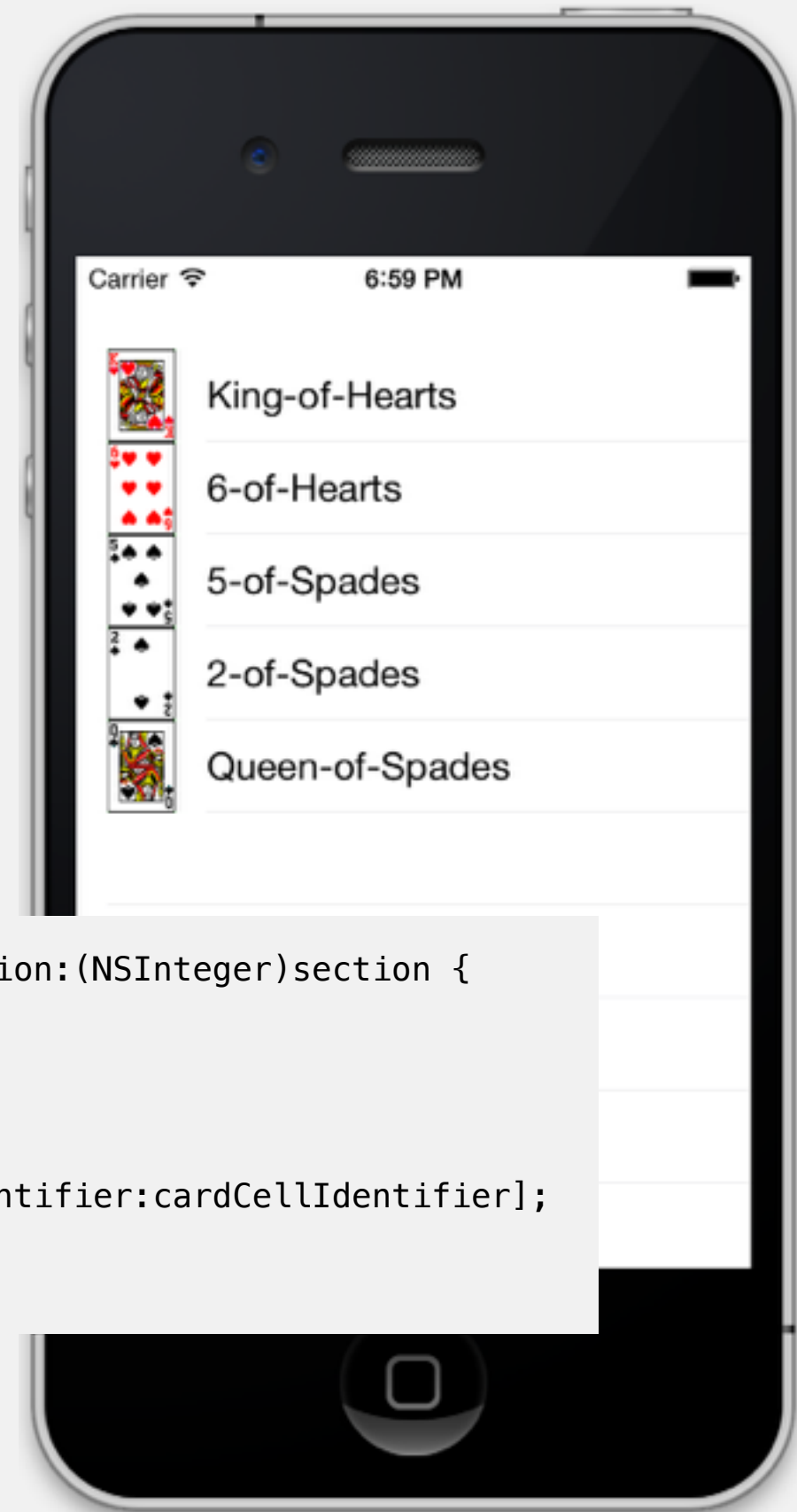
-(RMSGoFishPlayer *)player {
    if (!_player) {
        _player = self.game.players[0];
    }
    return _player;
}

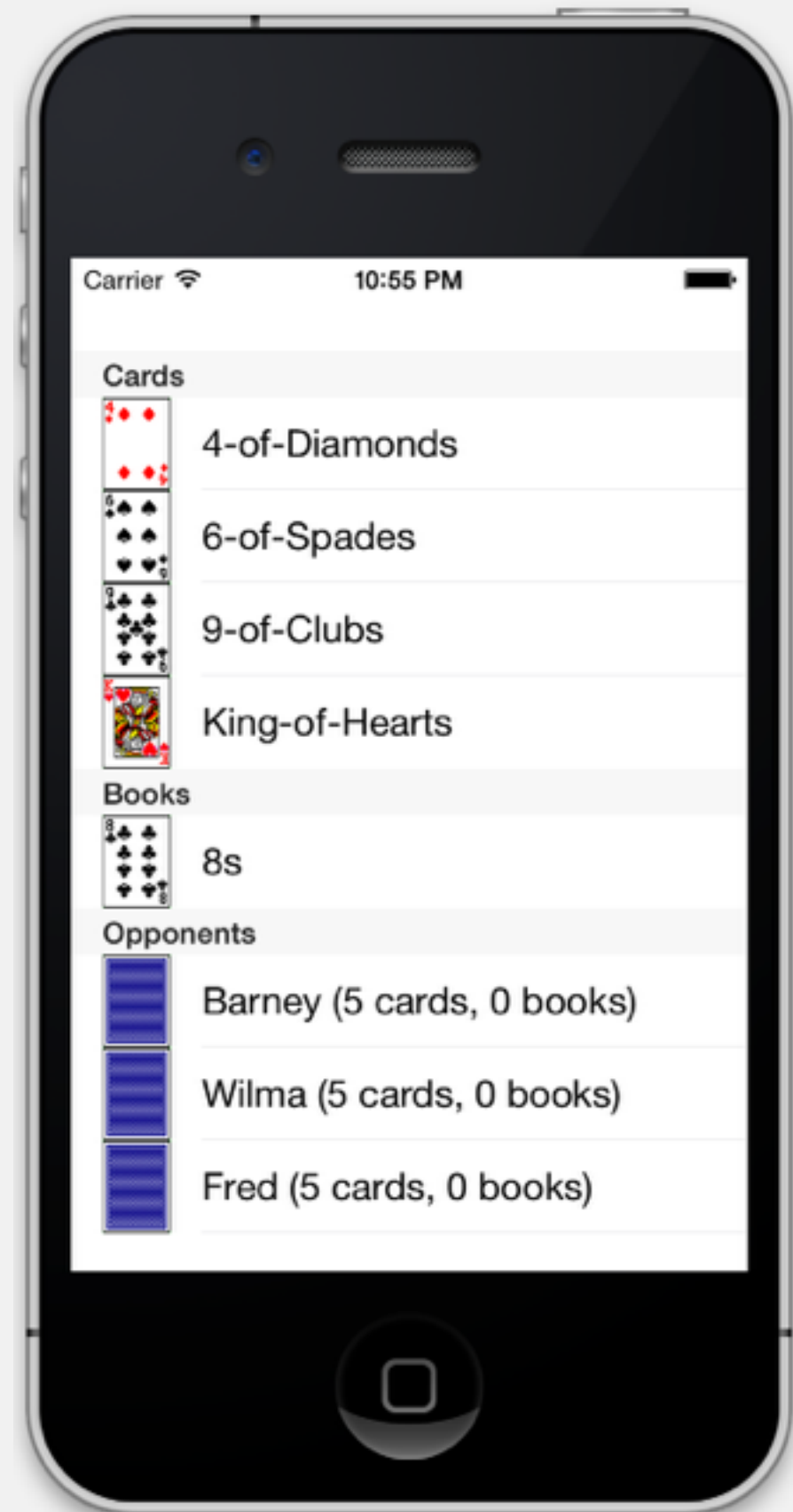
-(NSArray *)hand {
    return [self.player hand];
}

-(NSInteger)tableView:(UITableView *)tableView numberOfRowsInSectionInSection:(NSInteger)section {
    return [[self hand] count];
}

-(UITableViewCell *)tableView:(UITableView *)tableView
    cellForRowAtIndexPath:(NSIndexPath *)indexPath {
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:cardCellIdentifier];
    RMSPlayingCard *card = [self hand][indexPath.row];
    cell.textLabel.text = [card description];
    cell.imageView.image = [UIImage imageNamed:[card imageName]];
    return cell;
}

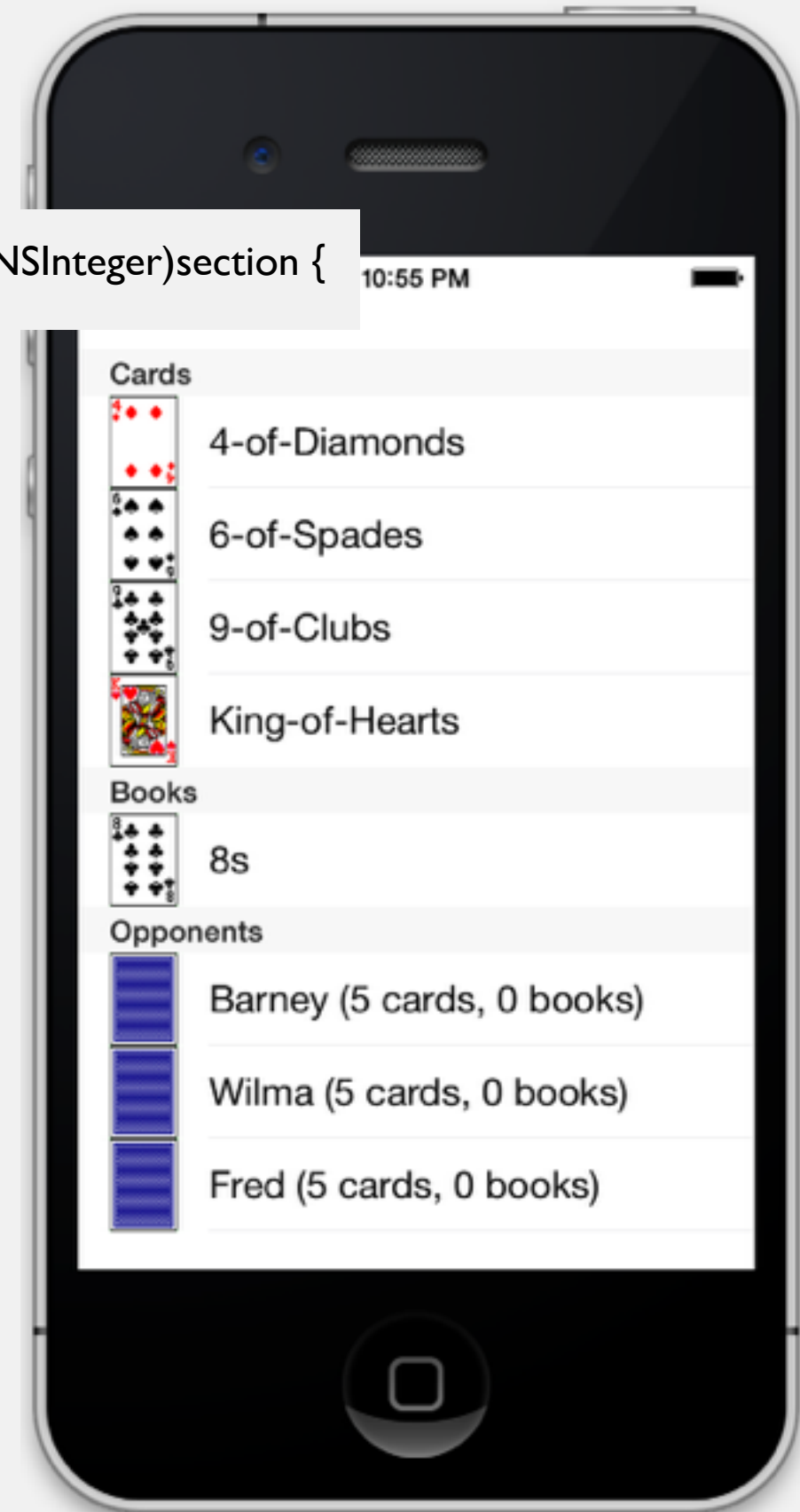
```





```
-(NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {  
    return 3;  
}
```

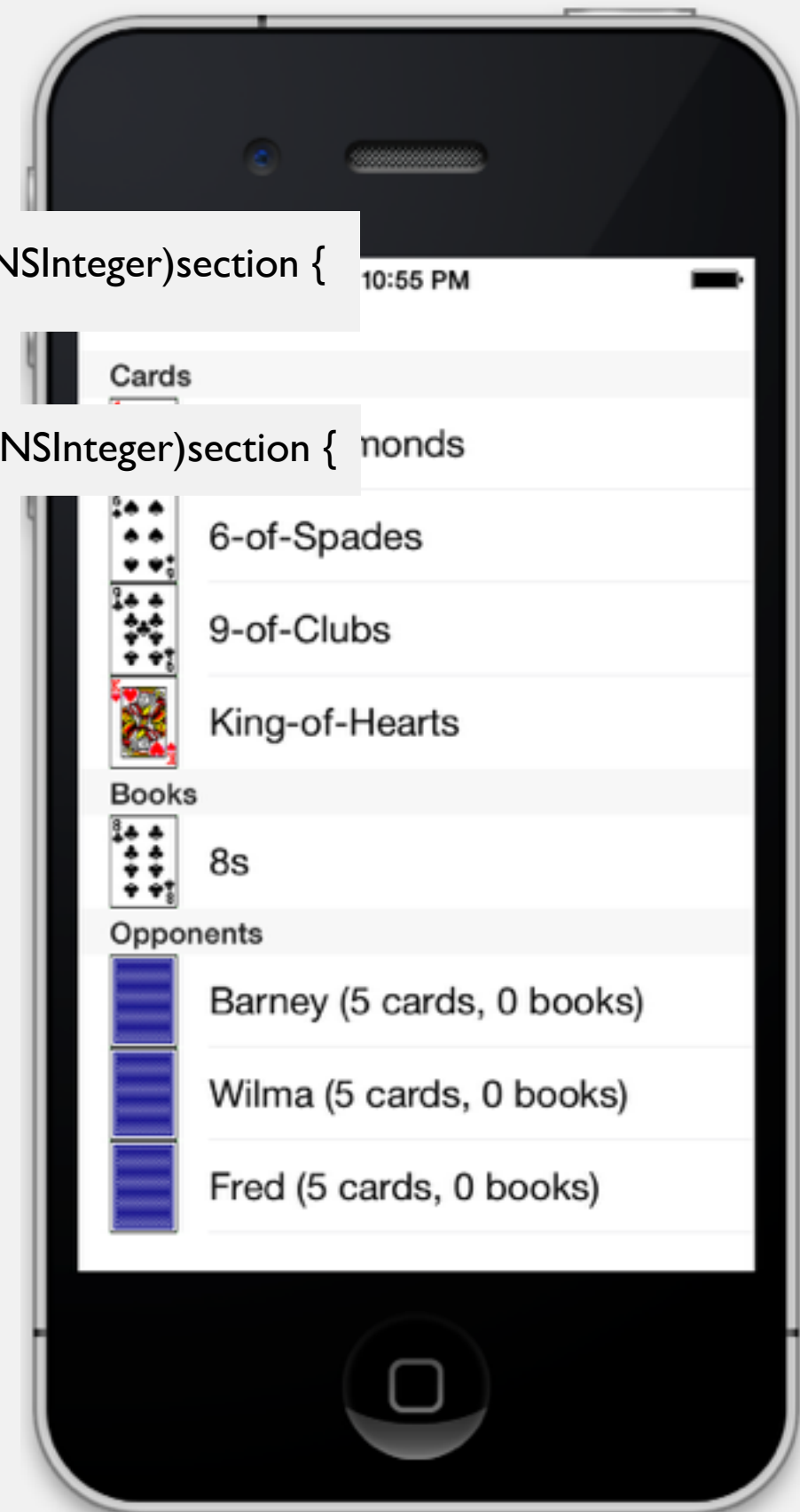
```
-(NSString *)tableView:(UITableView *)tableView titleForHeaderInSection:(NSInteger)section {  
    return @[@"Cards", @"Books", @"Opponents"][section];  
}
```



```
-(NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {  
    return 3;  
}
```

```
-(NSString *)tableView:(UITableView *)tableView titleForHeaderInSection:(NSInteger)section {  
    return @[@"Cards", @"Books", @"Opponents"][section];  
}
```

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



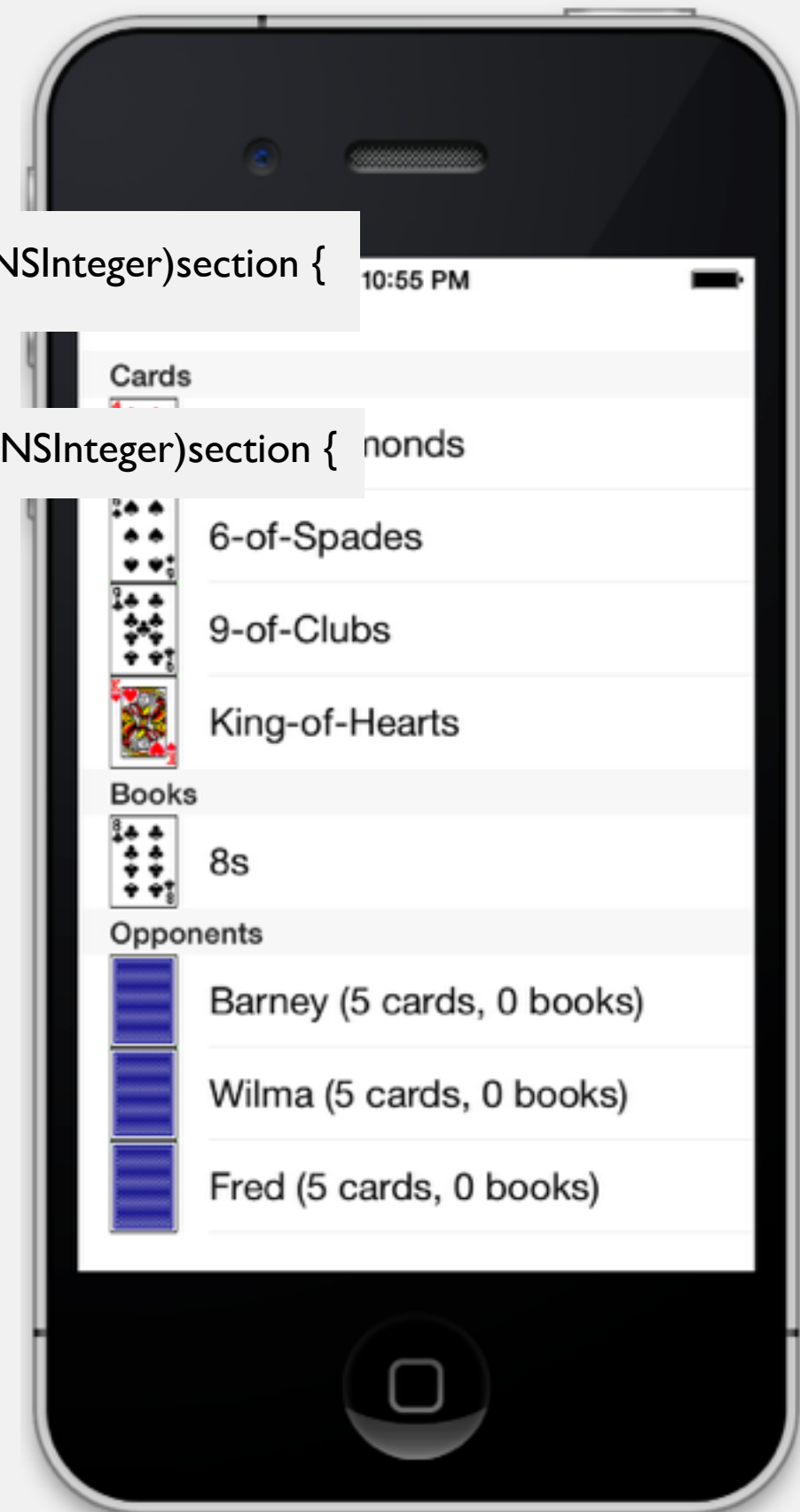
```

-(NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
    return 3;
}

-(NSString *)tableView:(UITableView *)tableView titleForHeaderInSection:(NSInteger)section {
    return @[@"Cards", @"Books", @"Opponents"][section];
}

-(NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section {
    switch (section) {
        case 0:
            return [[self hand] count];
        case 1:
            return [self.player.books count];
        case 2:
            return [[self opponents] count];
    }
    return 0;
}

```




```

-(NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
    return 3;
}

```

```

-(NSString *)tableView:(UITableView *)tableView titleForHeaderInSection:(NSInteger)section {
    return @[@"Cards", @"Books", @"Opponents"][section];
}

```

```

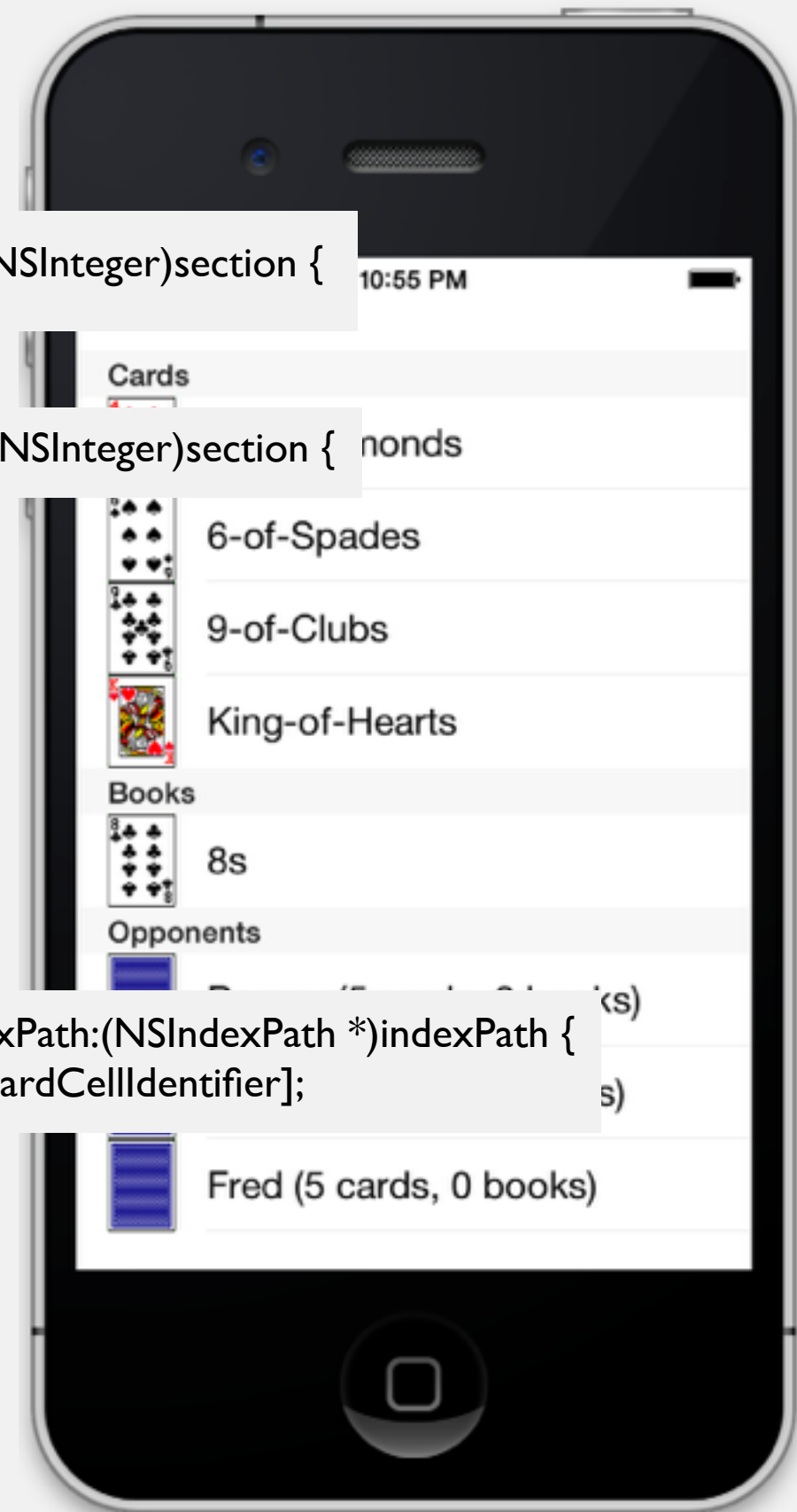
-(NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section {
    switch (section) {
        case 0:
            return [[self hand] count];
        case 1:
            return [self.player.books count];
        case 2:
            return [[self opponents] count];
    }
    return 0;
}

```

```

-(UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath {
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:cardCellIdentifier];
    RMSPlayingCard *card = [self hand][indexPath.row];
    cell.textLabel.text = [card description];
    cell.imageView.image = [UIImage imageNamed:[card imageName]];
    return cell;
}

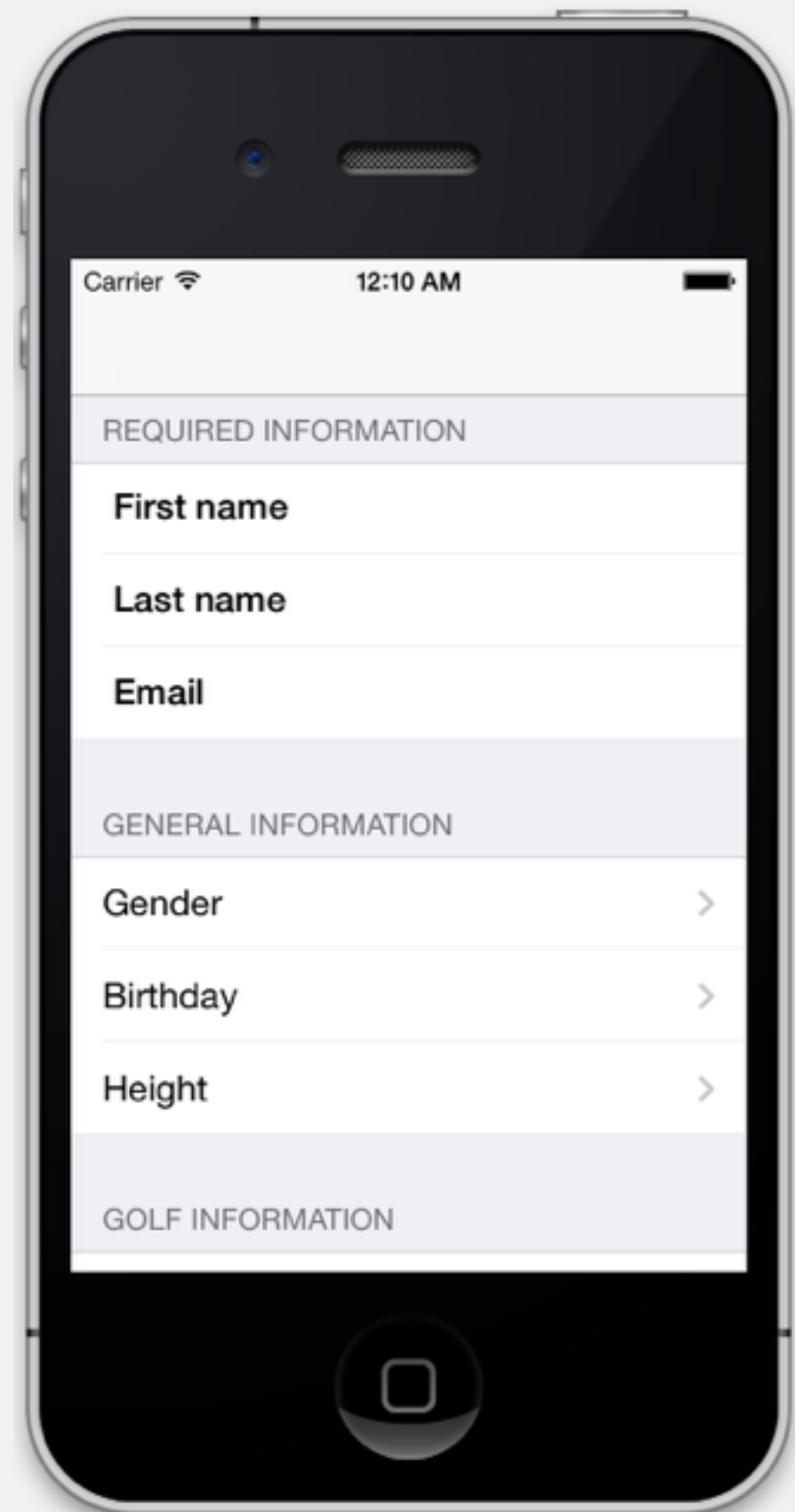
```



```

-(UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath {
    UITableViewCell *cell;
    switch (indexPath.section) {
        case 0: {
            cell = [tableView dequeueReusableCellWithIdentifier:cardCellIdentifier];
            RMSPlayingCard *card = [self hand][indexPath.row];
            cell.textLabel.text = [card description];
            cell.imageView.image = [UIImage imageNamed:[card imageName]];
            break;
        }
        case 1: {
            cell = [tableView dequeueReusableCellWithIdentifier:bookCellIdentifier];
            NSArray *book = self.player.books[indexPath.row];
            RMSPlayingCard *firstCard = [book firstObject];
            cell.textLabel.text = [NSString stringWithFormat:@"%s", firstCard.rank];
            cell.imageView.image = [UIImage imageNamed:[firstCard imageName]];
            break;
        }
        case 2: {
            cell = [tableView dequeueReusableCellWithIdentifier:opponentCellIdentifier];
            RMSGoFishPlayer *opponent = [self opponents][indexPath.row];
            cell.textLabel.text = [NSString stringWithFormat:@"%s (%d cards, %d books)", opponent.name, [opponent numberOfCards],
[[opponent books] count]];
            cell.imageView.image = [UIImage imageNamed:@"backs_blue"];
            break;
        }
        default:
            break;
    }
    return cell;
}

```



```
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath {
    static NSString *valueCellIdentifier = @"ValueCell";
    static NSString *buttonCellIdentifier = @"ButtonCell";
```

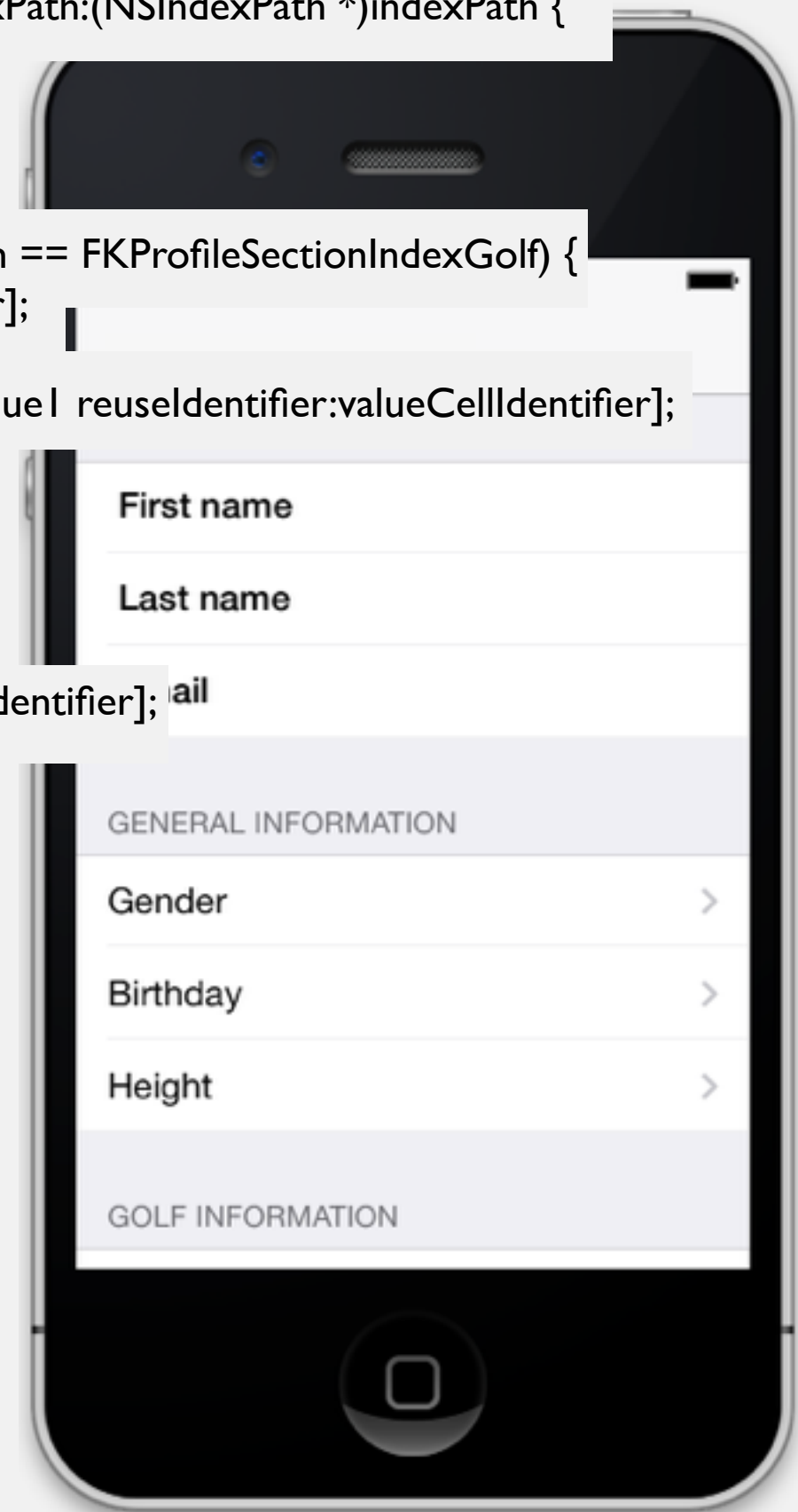
```
    UITableViewCell *cell = nil;
    if (indexPath.section == FKProfileSectionIndexGeneral || indexPath.section == FKProfileSectionIndexGolf) {
        cell = [tableView dequeueReusableCellWithIdentifier:valueCellIdentifier];
        if (cell == nil) {
            cell = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleValue1 reuseIdentifier:valueCellIdentifier];
            cell.accessoryType = UITableViewCellAccessoryDisclosureIndicator;
        }
    }
}
```

```
if (indexPath.section == FKProfileSectionIndexRequired) {
    cell = [tableView dequeueReusableCellWithIdentifier:FKTextEntryCellIdentifier];
    cell.selectionStyle = UITableViewCellSelectionStyleNone;
```

```
    UILabel *label = (UILabel *)[cell viewWithTag:10];
    UITextField *textField = (UITextField *)[cell viewWithTag:11];
    textField.secureTextEntry = NO;
    textField.keyboardType = UIKeyboardTypeDefault;
```

```
    UITableViewController *twin = self;
    if (indexPath.row == 0) {
        label.text = @"First name";
        textField.text = self.profile.firstname;
        textField.delegate = self.firstNameObserver;
    } else if (indexPath.row == 1) {
        label.text = @"Last name";
        textField.text = self.profile.lastname;
        textField.delegate = self.lastNameObserver;
```

```
    [self.firstNameObserver setNextTextField:textField withBlock:^(
        [twin.tableView scrollToRowAtIndexPath:[NSIndexPath indexPathForItem:indexPath.row inSection:indexPath.section]
            atScrollPosition:UITableViewScrollPositionBottom
            animated:YES];
    ]];
}
```



```
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath {
    static NSString *valueCellIdentifier = @"ValueCell";
    static NSString *buttonCellIdentifier = @"ButtonCell";

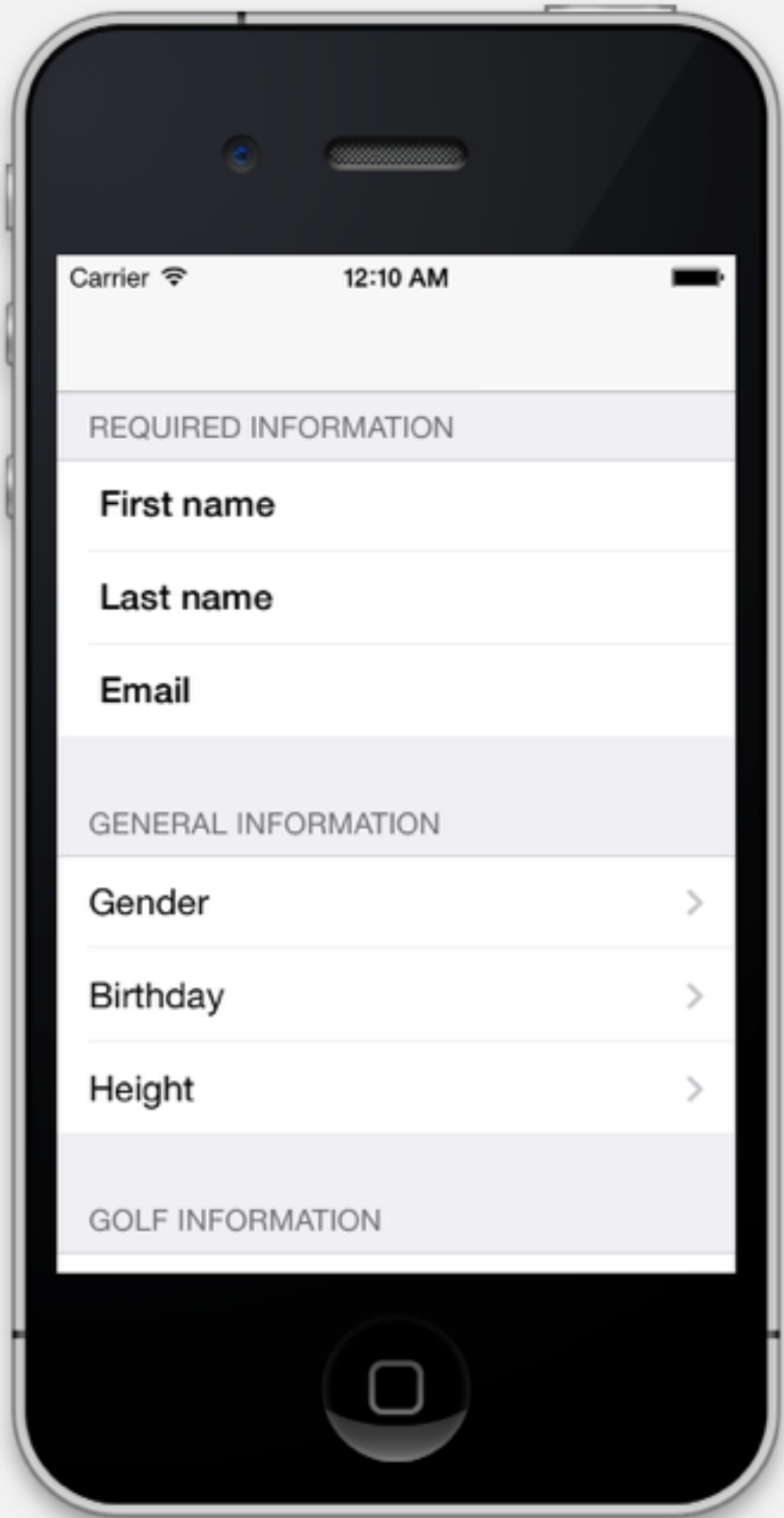
    UITableViewCell *cell = nil;
    if (indexPath.section == FKProfileSectionIndexGeneral || indexPath.section == FKProfileSectionIndexGolf) {
        cell = [tableView dequeueReusableCellWithIdentifier:valueCellIdentifier];
        if (cell == nil) {
            cell = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleValue1 reuseIdentifier:valueCellIdentifier];
            cell.accessoryType = UITableViewCellAccessoryDisclosureIndicator;
        }
    }

    if (indexPath.section == FKProfileSectionIndexRequired) {
        cell = [tableView dequeueReusableCellWithIdentifier:FKTextEntryCellIdentifier];
        cell.selectionStyle = UITableViewCellStyleNone;

        UILabel *label = (UILabel *)[cell viewWithTag:10];
        UITextField *textField = (UITextField *)[cell viewWithTag:11];
        textField.secureTextEntry = NO;
        textField.keyboardType = UIKeyboardTypeDefault;

        UITableViewController *twin = self;
        if (indexPath.row == 0) {
            label.text = @"First name";
            textField.text = self.profile.firstName;
            textField.delegate = self.firstNameObserver;
        } else if (indexPath.row == 1) {
            label.text = @"Last name";
            textField.text = self.profile.lastName;
            textField.delegate = self.lastNameObserver;
            [self.lastNameObserver setNextTextField:textField withBlock:^(
                [twin tableView scrollToRowAtIndexPath:[NSIndexPath indexPathForItem:indexPath.row inSection:indexPath.section]
                    atScrollPosition:UITableViewScrollPositionBottom
                        animated:YES];
            );
        }
    } else if (indexPath.row == 2) {
        label.text = @"Email";
        textField.text = self.profile.email;
        textField.delegate = self.emailObserver;
        textField.keyboardType = UIKeyboardTypeEmailAddress;
        [self.lastNameObserver setNextTextField:textField withBlock:^(
            [twin tableView scrollToRowAtIndexPath:[NSIndexPath indexPathForItem:indexPath.row inSection:indexPath.section]
                atScrollPosition:UITableViewScrollPositionBottom
                    animated:YES];
            );
    }
    } else if (indexPath.row == 3) {
        label.text = @"Password";
        textField.text = self.passwordOne;
        textField.secureTextEntry = YES;
        textField.delegate = self.passwordOneObserver;
        [self.emailObserver setNextTextField:textField withBlock:^(
            [twin tableView scrollToRowAtIndexPath:[NSIndexPath indexPathForItem:indexPath.row inSection:indexPath.section]
                atScrollPosition:UITableViewScrollPositionBottom
                    animated:YES];
            );
    }
    } else if (indexPath.row == 4) {
        label.text = @"Confirm";
        textField.text = self.passwordTwo;
        textField.secureTextEntry = YES;
        textField.delegate = self.passwordTwoObserver;
        [self.passwordOneObserver setNextTextField:textField withBlock:^(
            [twin tableView scrollToRowAtIndexPath:[NSIndexPath indexPathForItem:indexPath.row inSection:indexPath.section]
                atScrollPosition:UITableViewScrollPositionBottom
                    animated:YES];
            );
    }
    }
    }
    } else if (indexPath.section == FKProfileSectionIndexGeneral) {
        if (indexPath.row == 0) {
            cell.textLabel.text = @"Gender";
            cell.detailTextLabel.text = self.profile.gender;
        } else if (indexPath.row == 1) {
            cell.textLabel.text = @"Birthday";
            cell.detailTextLabel.text = [NSDateFormatter localizedStringFromDate:self.profile.birthday
                dateStyle:NSDateFormatterMediumStyle
                timeStyle:NSDateFormatterNoStyle];
        }
        } else if (indexPath.row == 2) {
            cell.textLabel.text = @"Height";
            if (self.profile.height != nil) {
                NSInteger height = [self.profile.height integerValue];
                cell.detailTextLabel.text = [NSString stringWithFormat:@"%d' %d\"", height / 12, height % 12];
            } else {
                cell.detailTextLabel.text = nil;
            }
        }
    }
    } else if (indexPath.section == FKProfileSectionIndexGolf) {
        if (indexPath.row == 0) {
            cell.accessoryType = UITableViewCellAccessoryDisclosureIndicator;
            cell.textLabel.text = @"Handicap";
            cell.detailTextLabel.text = [self.profile.handicap description];
        } else if (indexPath.row == 1) {
            cell.accessoryType = UITableViewCellAccessoryDisclosureIndicator;
            cell.textLabel.text = @"Stance";
            cell.detailTextLabel.text = self.profile.stance;
        }
    }
    } else if (indexPath.section == FKProfileSectionIndexSave || indexPath.section == FKProfileSectionIndexDelete) {
        cell = [tableView dequeueReusableCellWithIdentifier:buttonCellIdentifier];
        NSUInteger labelTag = 1919;
        if (cell == nil) {
            cell = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleValue1 reuseIdentifier:buttonCellIdentifier];
            UIView *contentView = [cell contentView];
            UILabel *label = [[UILabel alloc] initWithFrame:[contentView frame]];
            [label setAutoresizingMask:UIViewAutoresizingFlexibleWidth];
            label.tag = labelTag;
            UIFont *boldFont = [UIFont boldSystemFontOfSize:[UIFont buttonFontSize]];
            [label setFont:boldFont];
            [label setTextAlignment:NSTextAlignmentCenter];
            [label setBackgroundColor:[UIColor clearColor]];
            [contentView addSubview:label];
            label.text = @"Save";
        }
    }
    } if (indexPath.section == FKProfileSectionIndexSave) {
        cell.textLabel.textColor = [UIColor blackColor];
        if ([self.inRegistrationMode]) {
            UILabel *label = (UILabel *)[cell viewWithTag:labelTag];
            label.text = @"Join";
        }
    }

    if ([self.profileInformationIsValid]) {
        cell.selectionStyle = UITableViewCellSelectionStyleGray;
        [[UILabel *)[cell viewWithTag:labelTag] setTextColor:[UIColor whiteColor]];
        cell.backgroundColor = [UIColor darkGrayColor];
    }
    } else {
        cell.selectionStyle = UITableViewCellSelectionStyleNone;
        cell.backgroundColor = [UIColor lightGrayColor];
        [[UILabel *)[cell viewWithTag:labelTag] setTextColor:[UIColor grayColor]];
    }
    } else if (indexPath.section == FKProfileSectionIndexDelete) {
        UILabel *label = (UILabel *)[cell viewWithTag:labelTag];
        label.text = @"Delete";
        cell.selectionStyle = UITableViewCellSelectionStyleNone;
        [[UILabel *)[cell viewWithTag:labelTag] setTextColor:[UIColor whiteColor]];
        cell.backgroundColor = [UIColor redColor];
    }
    }
    }
    return cell;
}
```



```

@implementation RMSViewController

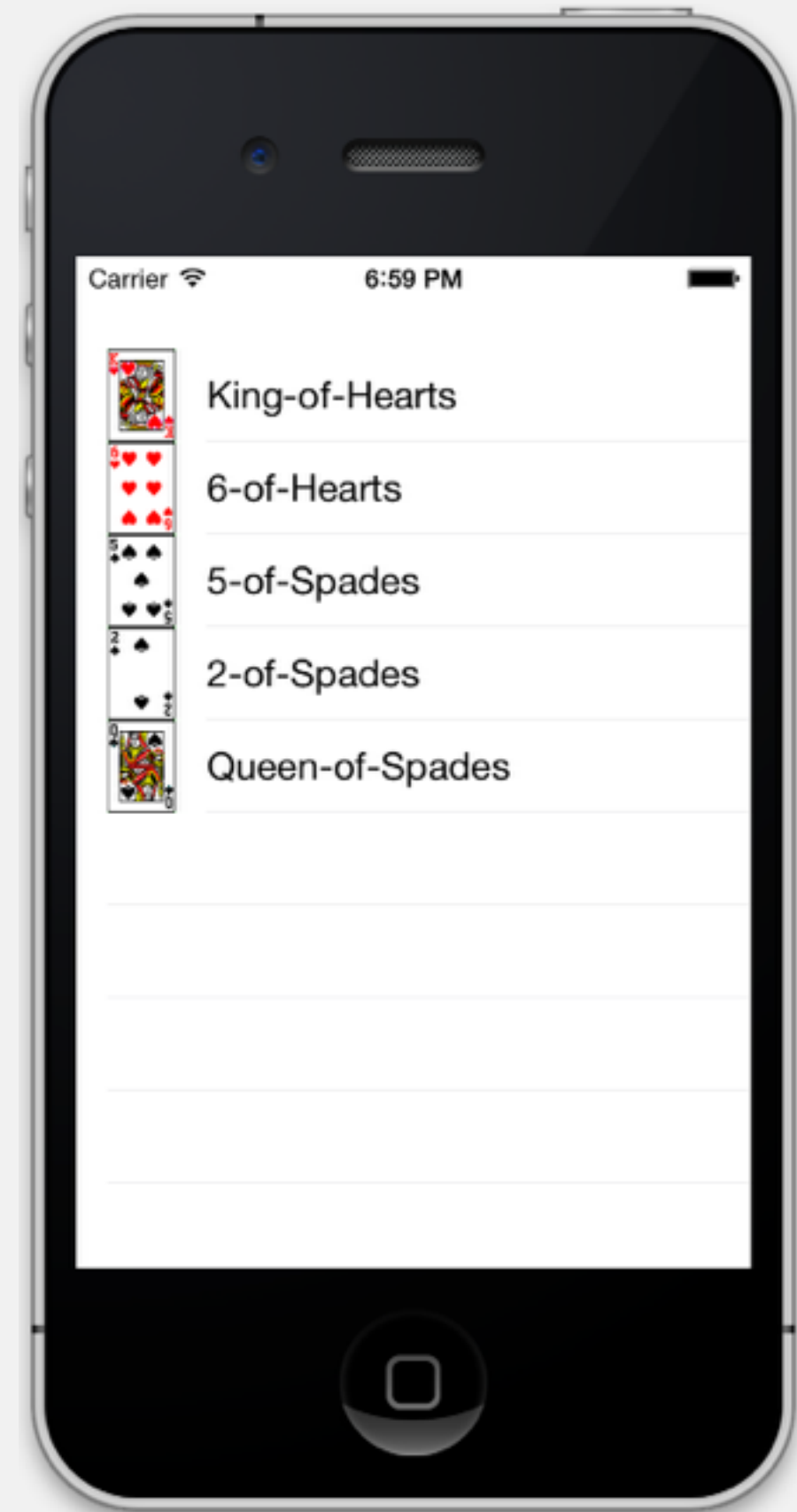
static NSString *cardCellIdentifier = @"CardCell";

-(RMSGoFishGame *)game {
    if (!_game) {
        _game = [RMSGoFishGame new];
        [_game deal];
    }
    return _game;
}

-(RMSGoFishPlayer *)player {
    if (!_player) {
        _player = self.game.players[0];
    }
    return _player;
}

-(NSArray *)hand {
    return [self.player hand];
}

```




```

@implementation RMSViewController

static NSString *cardCellIdentifier = @"CardCell";

-(RMSGoFishGame *)game {
    if (!_game) {
        _game = [RMSGoFishGame new];
        [_game deal];
    }
    return _game;
}

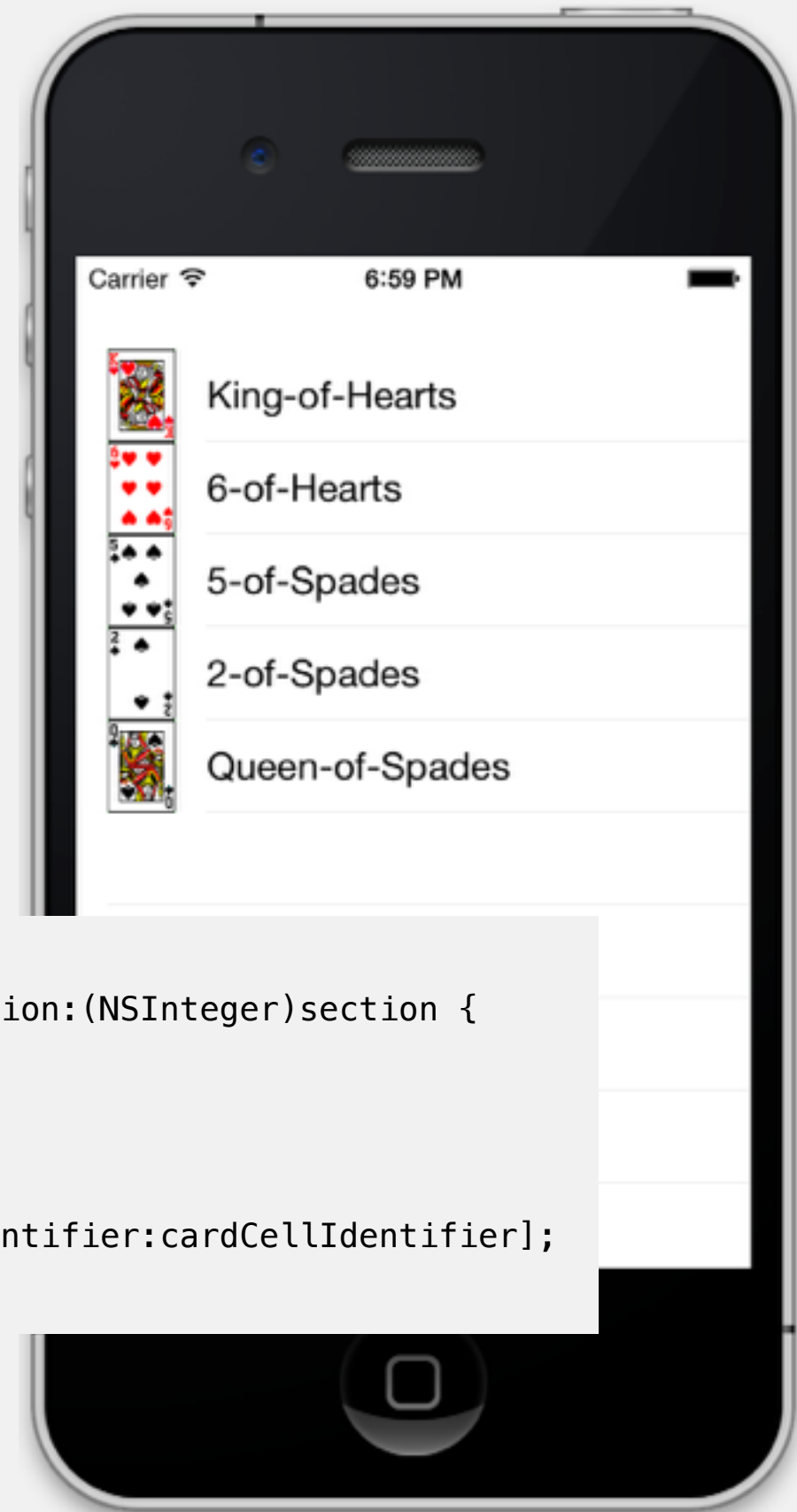
-(RMSGoFishPlayer *)player {
    if (!_player) {
        _player = self.game.players[0];
    }
    return _player;
}

-(NSArray *)hand {
    return [self.player hand];
}

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

-(UITableViewCell *)tableView:(UITableView *)tableView
    cellForRowAtIndexPath:(NSIndexPath *)indexPath {
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:cardCellIdentifier];
    RMSPlayingCard *card = [self hand][indexPath.row];
    cell.textLabel.text = [card description];
    cell.imageView.image = [UIImage imageNamed:[card imageName]];
    return cell;
}

```



```

@implementation RMSViewController

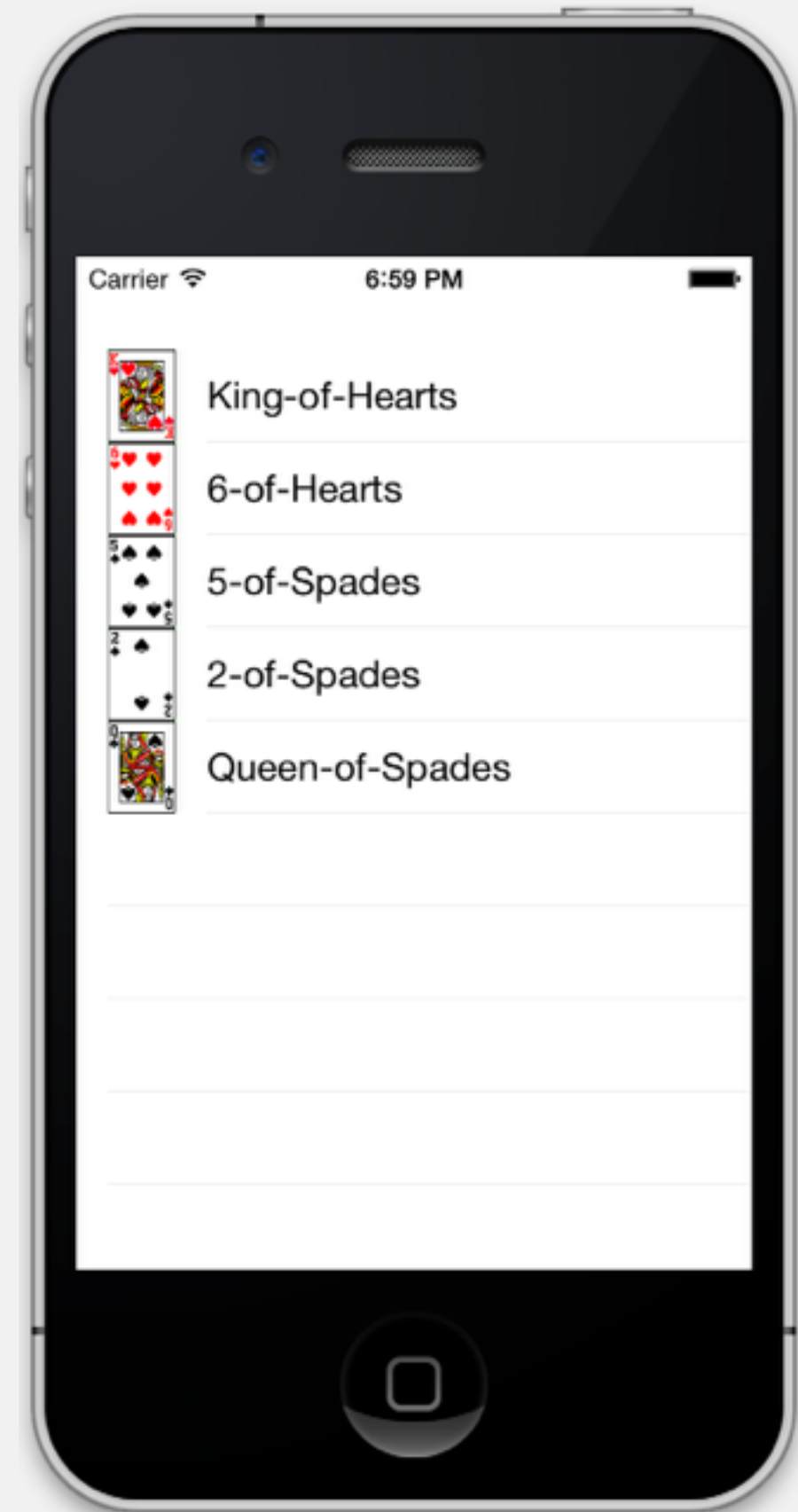
static NSString *cardCellIdentifier = @"CardCell";

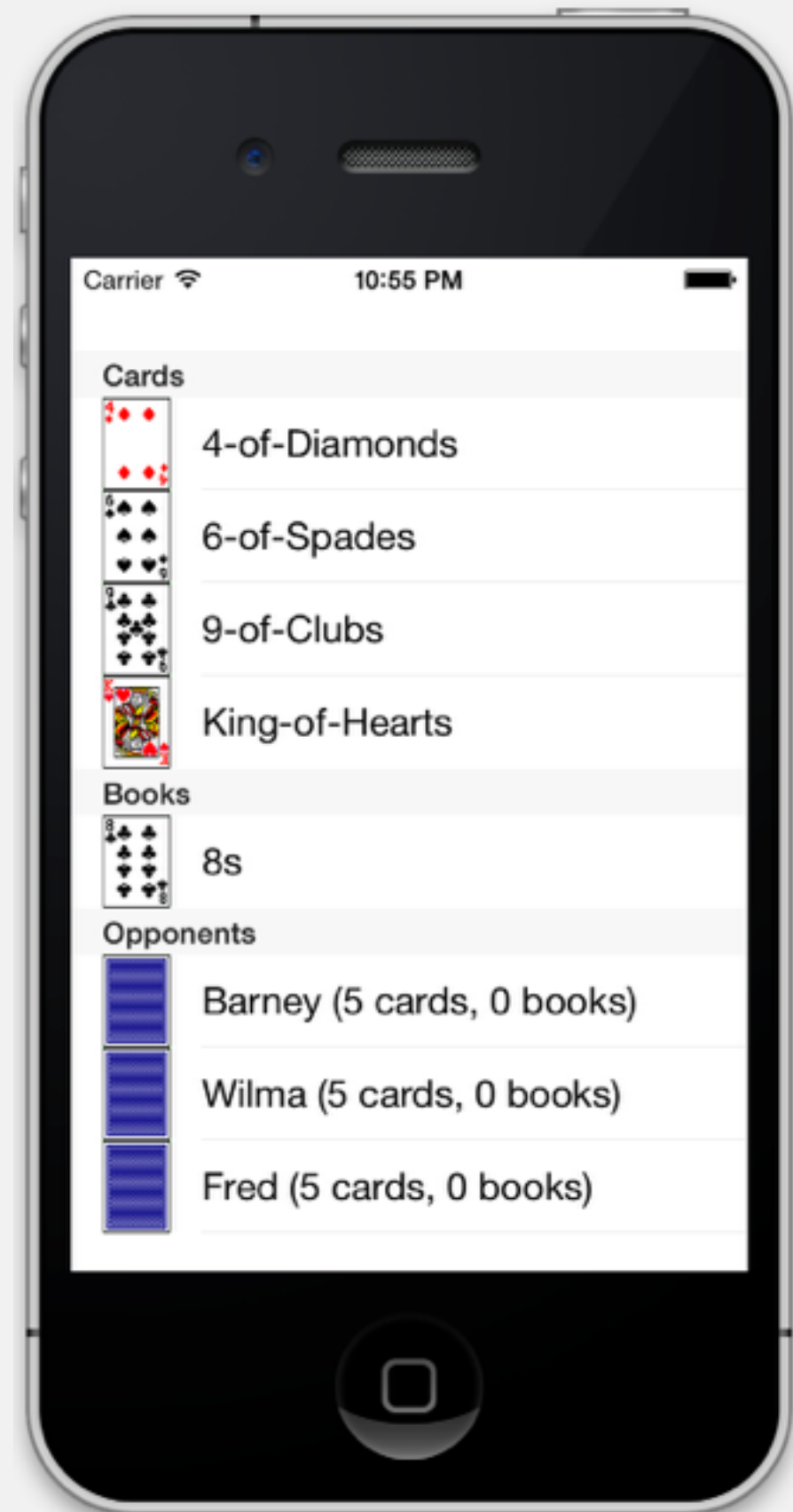
-(RMSGofishGame *)game {
    if (!_game) {
        _game = [RMSGofishGame new];
        [_game deal];
    }
    return _game;
}

-(RMSGofishPlayer *)player {
    if (!_player) {
        _player = self.game.players[0];
    }
    return _player;
}

-(NSArray *)hand {
    return [self.player hand];
}

```





@implementation RMSViewController

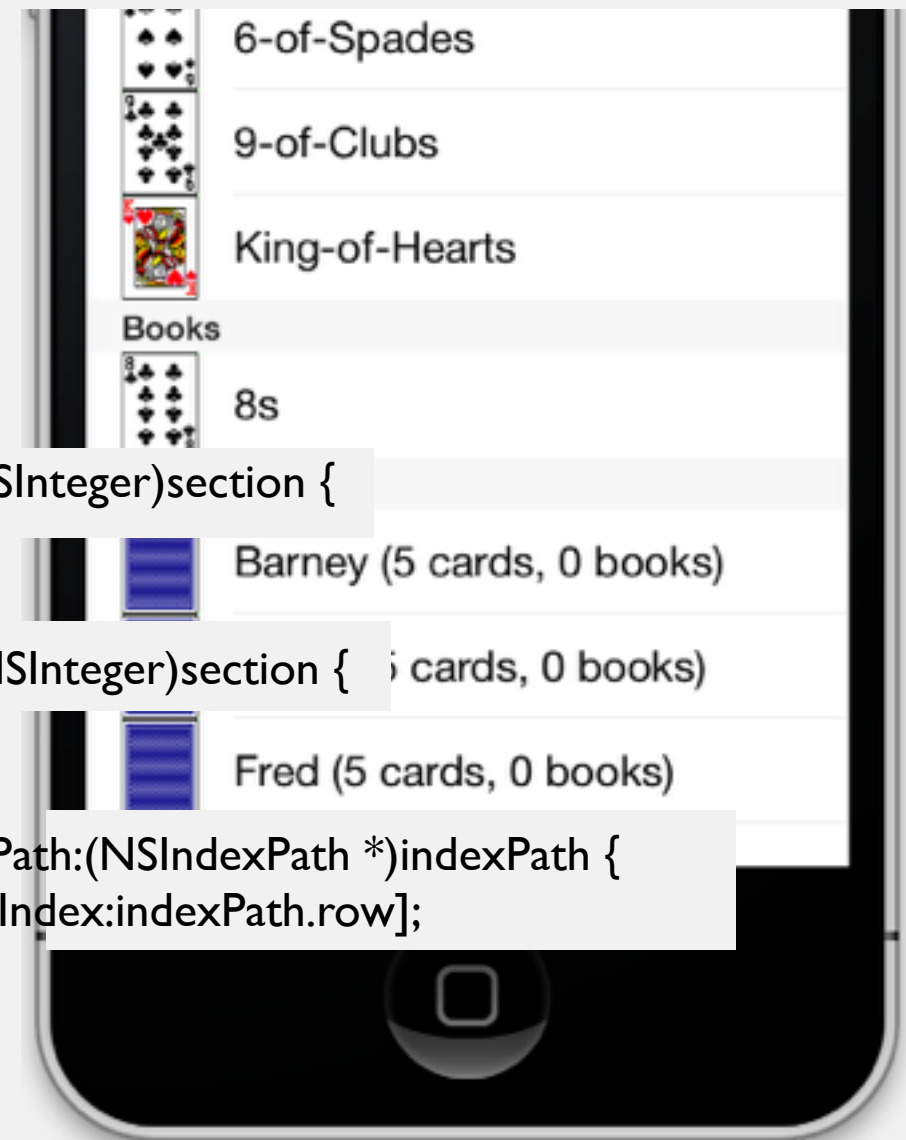
```
-(NSArray *)sections {
    if (!_sections) {
        RMSCardSection *cardSection = [[RMSCardSection alloc] initWithTitle:@"Cards" cellIdentifier:@"CardCell" player:self.player];
        RMSBookSection *bookSection = [[RMSBookSection alloc] initWithTitle:@"Books" cellIdentifier:@"BookCell" player:self.player];
        RMSOpponentsSection *opponentsSection = [[RMSOpponentsSection alloc] initWithTitle:@"Opponents"
cellIdentifier:@"OpponentCell" rows:[self opponents]];
        _sections = @[cardSection, bookSection, opponentsSection];
    }
    return _sections;
}

-(NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
    return [self.sections count];
}

-(NSString *)tableView:(UITableView *)tableView titleForHeaderInSection:(NSInteger)section {
    return [self.sections[section] titleForHeader];
}

-(NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section {
    return [self.sections[section] numberOfRows];
}

-(UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath {
    return [self.sections[indexPath.section] tableView:tableView cellForRowAtIndexPath:indexPath.row];
}
```



A Missing Section

```
@protocol RMTableViewSectionDelegate <NSObject>
- (NSString *)titleForHeader;
- (NSInteger)numberOfRows;
- (UITableViewCell *)tableView:(UITableView *)tableView
  cellForRowAtIndex:(NSUInteger)index;
@end
```

A Missing Section

```
@protocol RMTableViewSectionDelegate <NSObject>
- (NSString *)titleForHeader;
- (NSInteger)numberOfRows;
- (UITableViewCell *)tableView:(UITableView *)tableView
    cellForRowAtIndex:(NSUInteger)index;
@end
```

```
@interface RMSAbstractTableViewSection : NSObject<RMTableViewSectionDelegate>
@property (nonatomic) NSString *titleForHeader;
@property (nonatomic) NSString *cellIdentifier;
-(instancetype)initWithTitle:(NSString *)title cellIdentifier:(NSString *)cellIdentifier;
-(NSArray *)rows;
-(NSString *)textForRowAtIndex:(NSUInteger)index;
-(UIImage *)imageForRowAtIndex:(NSUInteger)index;
@end
```

A Missing Section

```
@protocol RMSTableViewSectionDelegate <NSObject>
- (NSString *)titleForHeader;
- (NSInteger)numberOfRows;
- (UITableViewCell *)tableView:(UITableView *)tableView
    cellForRowAtIndex:(NSUInteger)index;
@end
```

```
@interface RMSAbstractTableViewSection : NSObject<RMSTableViewSectionDelegate>
@property (nonatomic) NSString *titleForHeader;
@property (nonatomic) NSString *cellIdentifier;
-(instancetype)initWithTitle:(NSString *)title cellIdentifier:(NSString *)cellIdentifier
-(NSArray *)rows;
-(NSString *)textForRowAtIndex:(NSUInteger)index;
-(UIImage *)imageForRowAtIndex:(NSUInteger)index;
@end
```

```
@interface RMSGenericTableViewSection : RMSAbstractTableViewSection
@property (readonly) NSArray *rows;
-(instancetype)initWithTitle:(NSString *)title
    cellIdentifier:(NSString *)cellIdentifier rows:(NSArray *)rows;
@end
```

```

@implementation RMSAbstractTableViewSection
static NSString *genericCellIdentifier = @"Cell";

-(instancetype)initWithTitle:(NSString *)title cellIdentifier:(NSString *)cellIdentifier {
    self = [super init];
    if (self) {
        _titleForHeader = title;
        _cellIdentifier = cellIdentifier;
    }
    return self;
}

-(NSString *)cellIdentifier {
    if (!_cellIdentifier) {
        _cellIdentifier = genericCellIdentifier;
    }
    return _cellIdentifier;
}

-(NSInteger)numberOfRows {
    return [[self rows] count];
}

-(UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSUInteger)index {
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:[self cellIdentifier]];
    cell.textLabel.text = [self textForRowAtIndex:index];
    cell.imageView.image = [self imageForRowAtIndex:index];
    return cell;
}

@end

```

```

@implementation RMSAbstractTableViewSection
static NSString *genericCellIdentifier = @"Cell";

-(instancetype)initWithTitle:(NSString *)title cellIdentifier:(NSString *)cellIdentifier {
    self = [super init];
    if (self) {
        _titleLabel = title;
        _cellIdentifier = cellIdentifier;
    }
    return self;
}

-(NSString *)cellIdentifier {
    if (!_cellIdentifier) {
        _cellIdentifier = genericCellIdentifier;
    }
    return _cellIdentifier;
}

-(NSInteger)numberOfRows {
    return [[self rows] count];
}

-(UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSUInteger)index {
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:[self cellIdentifier]];
    cell.textLabel.text = [self textForRowAtIndexPath:index];
    cell.imageView.image = [self imageForRowAtIndexPath:index];
    return cell;
}

-(NSArray *)rows {
    return @[];
}

-(NSString *)textForRowAtIndexPath:(NSUInteger)index {
    return [[self rows][index] description];
}

-(UIImage *)imageForRowAtIndexPath:(NSUInteger)index {
    return nil;
}

@end

```

A Common Section

A Common Section

```
@implementation RMSGenericTableViewSection
```

```
-(instancetype)initWithTitle:(NSString *)title cellIdentifier:(NSString *)cellIdentifier  
rows:(NSArray *)rows {  
    self = [super initWithTitle:title cellIdentifier:cellIdentifier];  
    if (self) {  
        _rows = rows;  
    }  
    return self;  
}
```

```
@end
```

@implementation RMSViewController

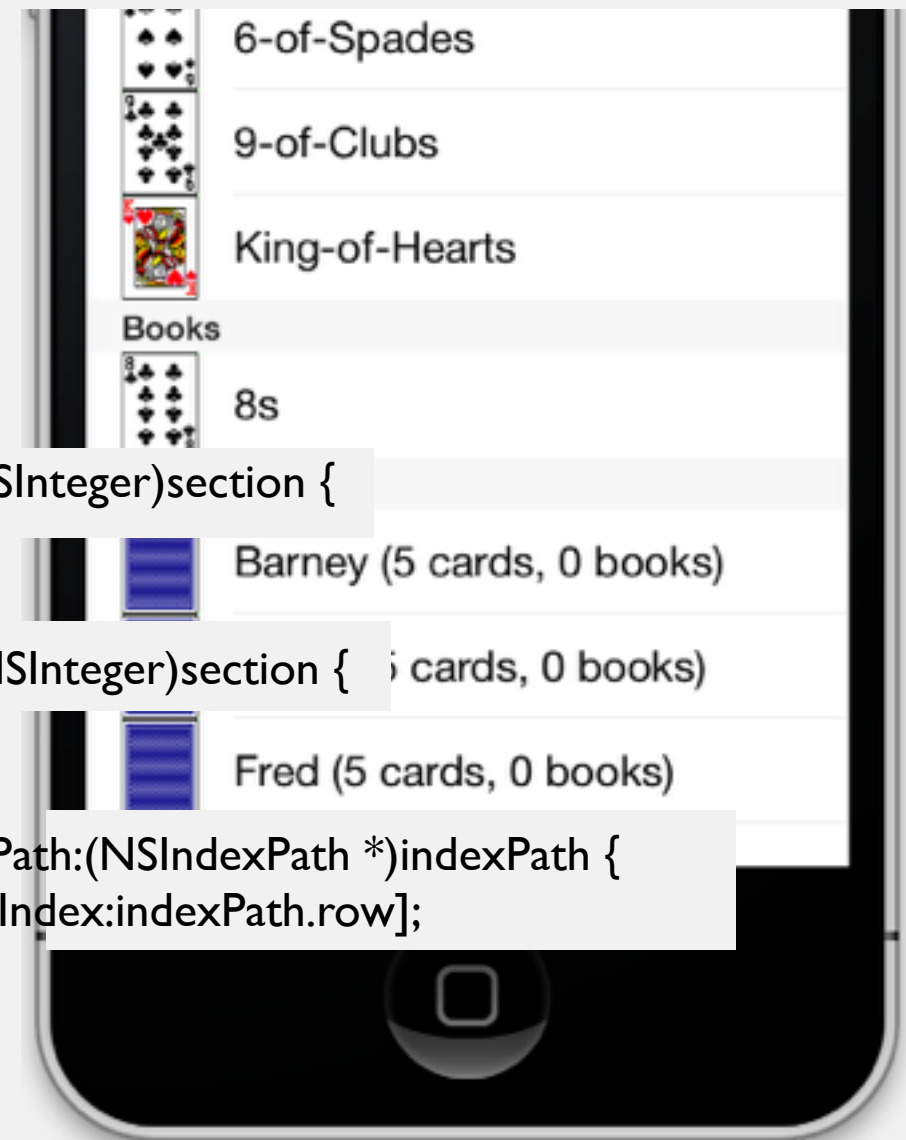
```
-(NSArray *)sections {
    if (!_sections) {
        RMSCardSection *cardSection = [[RMSCardSection alloc] initWithTitle:@"Cards" cellIdentifier:@"CardCell" player:self.player];
        RMSBookSection *bookSection = [[RMSBookSection alloc] initWithTitle:@"Books" cellIdentifier:@"BookCell" player:self.player];
        RMSOpponentsSection *opponentsSection = [[RMSOpponentsSection alloc] initWithTitle:@"Opponents"
cellIdentifier:@"OpponentCell" rows:[self opponents]];
        _sections = @[cardSection, bookSection, opponentsSection];
    }
    return _sections;
}

-(NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
    return [self.sections count];
}

-(NSString *)tableView:(UITableView *)tableView titleForHeaderInSection:(NSInteger)section {
    return [self.sections[section] titleForHeader];
}

-(NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section {
    return [self.sections[section] numberOfRows];
}

-(UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath {
    return [self.sections[indexPath.section] tableView:tableView cellForRowAtIndexPath:indexPath.row];
}
```



```
@interface RMSCardSection : RMSAbstractTableViewSection
@property RMSGoFishPlayer *player;
-(instancetype)initWithTitle:(NSString *)title cellIdentifier:(NSString *)cellIdentifier
player:(RMSGoFishPlayer *)player;
@end

@implementation RMSCardSection
@synthesize player = _player;

-(instancetype)initWithTitle:(NSString *)title cellIdentifier:(NSString *)cellIdentifier
player:(RMSGoFishPlayer *)player {
    self = [super initWithTitle:title cellIdentifier:cellIdentifier];
    if (self) {
        _player = player;
    };
    return self;
}

-(NSArray *)rows {
    return [self.player hand];
}

-(UIImage *)imageForRowAtIndex:(NSUInteger)index {
    RMSPlayingCard *card = [self rows][index];
    return [UIImage imageNamed:[card imageName]];
}

@end
```

```
@interface RMSBookSection : RMSAbstractTableViewSection
@property RMSGoFishPlayer *player;
-(instancetype)initWithTitle:(NSString *)title cellIdentifier:(NSString *)cellIdentifier player:
(RMSGoFishPlayer *)player;
@end

@implementation RMSBookSection

-(instancetype)initWithTitle:(NSString *)title cellIdentifier:(NSString *)cellIdentifier player:
(RMSGoFishPlayer *)player {
    self = [super initWithTitle:title cellIdentifier:cellIdentifier];
    if (self) {
        _player = player;
    };
    return self;
}

-(NSArray *)rows {
    return [self.player books];
}

-(UIImage *)imageForRowAtIndex:(NSUInteger)index {
    RMSPlayingCard *card = [self rows][index][0];
    return [UIImage imageNamed:[card imageName]];
}

-(NSString *)textForRowAtIndex:(NSUInteger)index {
    RMSPlayingCard *card = [self rows][index][0];
    return [NSString stringWithFormat:@"%s", card.rank];
}

@end
```

```
@interface RMSOpponentsSection : RMSGenericTableViewSection
@end

@implementation RMSOpponentsSection

-(UIImage *)imageForRowAtIndex:(NSUInteger)index {
    return [UIImage imageNamed:@"backs_blue"];
}

-(NSString *)textForRowAtIndex:(NSUInteger)index {
    RMSGoFishPlayer *opponent = [self rows][index];
    return [NSString stringWithFormat:@"%@" (%d cards, %d books)",
        opponent.name,
        [opponent numberOfCards],
        [opponent books] count]];
}

@end
```

So What About Handling Selections?

RMSTableViews

<https://github.com/RoleModel/RMSTableViews>

An Improved UITableViewController

```
@interface RMSTableViewController : UITableViewController
@property (nonatomic, strong, readonly) NSArray *sections;
- (NSArray *)generateSections;
- (void)sectionGenerationDidComplete;
@end
```

An Improved UITableViewController

```
@interface RMTableViewController : UITableViewController
@property (nonatomic, strong, readonly) NSArray *sections;
- (NSArray *)generateSections;
- (void)sectionGenerationDidComplete;
@end
```

```
@interface RMTableSection : NSObject
@property (nonatomic, strong) NSString *headerTitle;
@property (nonatomic, strong) NSString *footerTitle;
- (NSInteger)rowCount;
- (UITableViewCell *)cellForIndex:(NSInteger)index;
@end
```


An Improved UITableViewController

```
@interface RMSTableViewController : UITableViewController
@property (nonatomic, strong, readonly) NSArray *sections;
- (NSArray *)generateSections;
- (void)sectionGenerationDidComplete;
@end
```

```
@interface RMSTableViewSection : NSObject
@property (nonatomic, strong) NSString *headerTitle;
@property (nonatomic, strong) NSString *footerTitle;
- (NSInteger)rowCount;
- (UITableViewCell *)cellForIndex:(NSInteger)index;
@end
```

```
@protocol RMSTableViewCell <NSObject>
@optional
- (void)respondToSelection;
@end
```

```
@implementation RMSTableViewController
```

```
- (NSArray *)generateSections {  
    return @[];  
}  
  
- (void)sectionGenerationDidComplete {  
}  
  
- (NSArray *)sections {  
    if (_sections == nil) {  
        _sections = [self generateSections];  
        [self sectionGenerationDidComplete];  
    }  
    return _sections;  
}  
  
- (void)tableView:(UITableView *)tableView  
    didSelectRowAtIndexPath:(NSIndexPath *)indexPath {  
    [tableView deselectRowAtIndexPath:indexPath animated:YES];  
    id cell = [tableView cellForRowAtIndexPath:indexPath];  
    if ([cell respondsToSelector:@selector(respondToSelection)]) {  
        [cell respondToSelection];  
    }  
}  
...  
}
```

```

- (UITableViewCell *)tableView:(UITableView *)tableView
  cellForRowAtIndexPath:(NSIndexPath *)indexPath {
    return [self.sections[indexPath.section] cellForIndex:indexPath.row];
}

- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
    return [self.sections count];
}

- (NSInteger)tableView:(UITableView *)tableView
  numberOfRowsInSection:(NSInteger)sectionIndex {
    return [self.sections[sectionIndex] rowCount];
}

- (NSString *)tableView:(UITableView *)tableView titleForHeaderInSection:
  (NSInteger)sectionIndex {
    RMSTableViewSection *section = self.sections[sectionIndex];
    return section.headerTitle;
}

- (NSString *)tableView:(UITableView *)tableView titleForFooterInSection:
  (NSInteger)sectionIndex {
    RMSTableViewSection *section = self.sections[sectionIndex];
    return section.footerTitle;
}

```

@end

```
@implementation RMSTableViewSection
```

```
- (NSInteger)rowCount {  
    @throw [NSEException exceptionWithName:NSGenericException  
        reason:@"Subclasses must implement rowCount"  
        userInfo:nil];  
}  
  
- (UITableViewCell *)cellForIndex:(NSInteger)index {  
    @throw [NSEException exceptionWithName:NSGenericException  
        reason:@"Subclasses must implement cellForIndex:"  
        userInfo:nil];  
}
```

```
@end
```

Sections Were Missing, but...

- In a dynamic table, they just about always do the same thing
 - Identify the represented objects
 - Provide the cell
- The cell is unique
- Don't put the intelligence in the Section, but in the Cell

```

@interface RMSDynamicSection : RMSTableViewSection
@property (nonatomic, strong) UITableView *tableView;
@property (nonatomic, strong) NSString *cellIdentifier;
@property (nonatomic, strong) NSArray *representedObjects;
- (id)initWithTableView:(UITableView *)tableView cellIdentifier:(NSString *)cellIdentifier;
@end

@implementation RMSDynamicSection

- (id)initWithTableView:(UITableView *)tableView
    cellIdentifier:(NSString *)cellIdentifier {
    self = [super init];
    if (self) {
        _tableView = tableView;
        _cellIdentifier = cellIdentifier;
    }
    return self;
}

- (NSInteger)rowCount {
    return [self.representedObjects count];
}

- (UITableViewCell *)cellForIndex:(NSInteger)index {
    id cell = [self.tableView dequeueReusableCellWithIdentifier:self.cellIdentifier];
    if ([cell respondsToSelector:@selector(bindObject:)]) {
        [cell bindObject:self.representedObjects[index]];
    }
    return cell;
}
@end

```

```
@implementation RMSWordCell  
  
- (void)bindObject:(id)object {  
    self.textLabel.text = object;  
}  
  
@end
```

```
@implementation RMSWordCell
```

```
- (void)bindObject:(id)object {  
    self.textLabel.text = object;  
}
```

```
@end
```

```
@implementation RMSCardCell
```

```
- (void)bindObject:(id)object {  
    RMSPlayingCard *card = object;  
    self.textLabel.text = [card description];  
    self.imageView.image = [UIImage imageNamed:[card imageName]];  
}
```

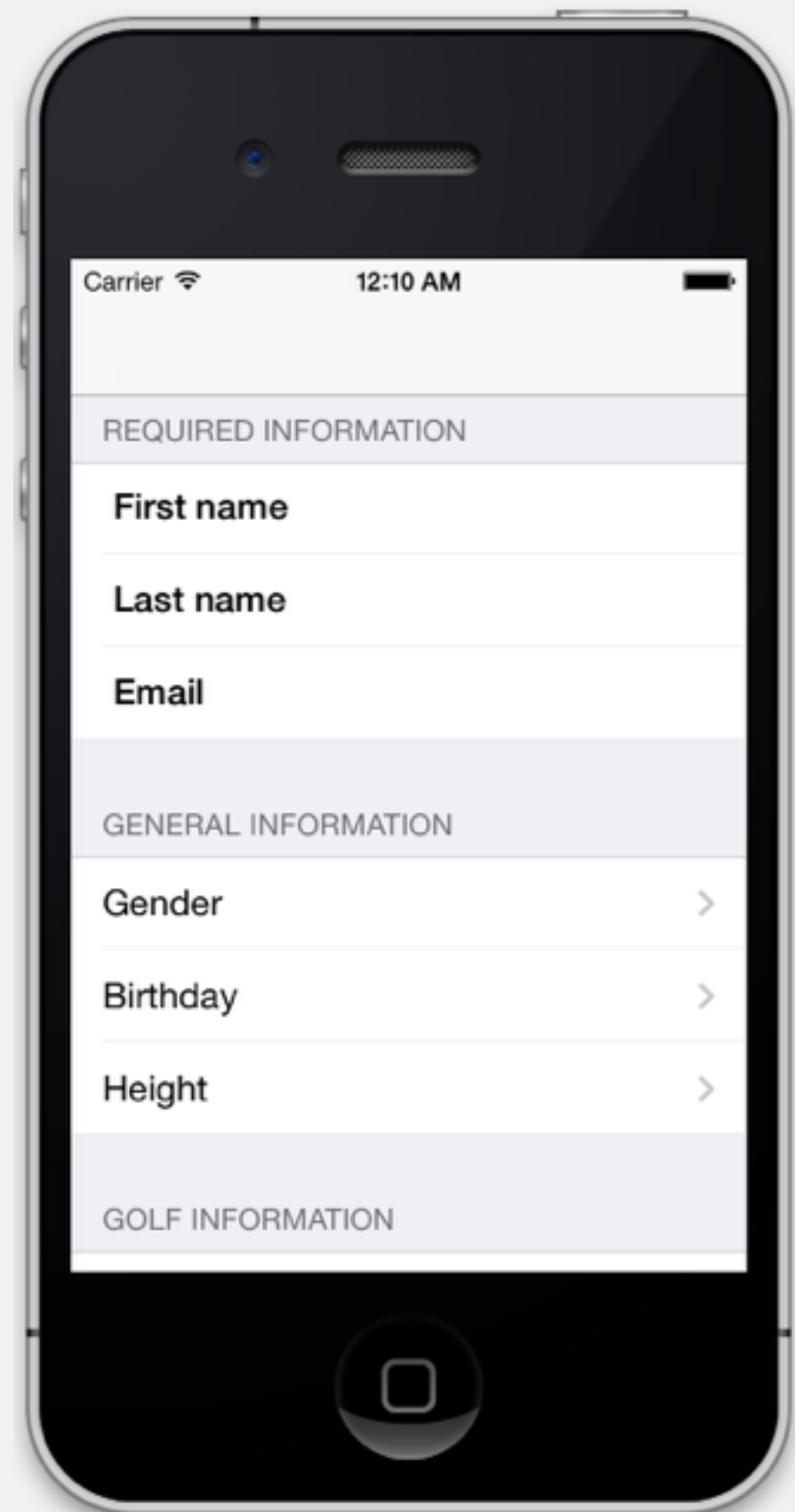
```
@end
```


So What About The “Form View”?

Data editing, Settings views...

The Form Descriptor

- Defines the structure of a form
- Can be expressed in code, as a Plist, or JSON
- Consists of an array of section dictionaries
- Section dictionaries specify section properties and contain an array of “row” dictionaries
- Row dictionaries describe cells



```
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath {
    static NSString *valueCellIdentifier = @"ValueCell";
    static NSString *buttonCellIdentifier = @"ButtonCell";

    UITableViewCell *cell = nil;
    if (indexPath.section == FKProfileSectionIndexGeneral || indexPath.section == FKProfileSectionIndexGolf) {
        cell = [tableView dequeueReusableCellWithIdentifier:valueCellIdentifier];
        if (cell == nil) {
            cell = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleValue1 reuseIdentifier:valueCellIdentifier];
            cell.accessoryType = UITableViewCellAccessoryDisclosureIndicator;
        }
    }

    if (indexPath.section == FKProfileSectionIndexRequired) {
        cell = [tableView dequeueReusableCellWithIdentifier:FKTextEntryCellIdentifier];
        cell.selectionStyle = UITableViewCellStyleNone;

        UILabel *label = (UILabel *)[cell viewWithTag:10];
        UITextField *textField = (UITextField *)[cell viewWithTag:11];
        textField.secureTextEntry = NO;
        textField.keyboardType = UIKeyboardTypeDefault;

        UITableViewController *twin = self;
        if (indexPath.row == 0) {
            label.text = @"First name";
            textField.text = self.profile.firstName;
            textField.delegate = self.firstNameObserver;
        } else if (indexPath.row == 1) {
            label.text = @"Last name";
            textField.text = self.profile.lastName;
            textField.delegate = self.lastNameObserver;
            [self.lastNameObserver setNextTextField:textField withBlock:^(
                [twin tableView scrollToRowAtIndexPath:[NSIndexPath indexPathForItem:indexPath.row inSection:indexPath.section]
                    atScrollPosition:UITableViewScrollPositionBottom
                    animated:YES];
            );
        }
    } else if (indexPath.row == 2) {
        label.text = @"Email";
        textField.text = self.profile.email;
        textField.delegate = self.emailObserver;
        textField.keyboardType = UIKeyboardTypeEmailAddress;
        [self.lastNameObserver setNextTextField:textField withBlock:^(
            [twin tableView scrollToRowAtIndexPath:[NSIndexPath indexPathForItem:indexPath.row inSection:indexPath.section]
                atScrollPosition:UITableViewScrollPositionBottom
                animated:YES];
            );
    }
    } else if (indexPath.row == 3) {
        label.text = @"Password";
        textField.text = self.passwordOne;
        textField.secureTextEntry = YES;
        textField.delegate = self.passwordOneObserver;
        [self.emailObserver setNextTextField:textField withBlock:^(
            [twin tableView scrollToRowAtIndexPath:[NSIndexPath indexPathForItem:indexPath.row inSection:indexPath.section]
                atScrollPosition:UITableViewScrollPositionBottom
                animated:YES];
            );
    }
    } else if (indexPath.row == 4) {
        label.text = @"Confirm";
        textField.text = self.passwordTwo;
        textField.secureTextEntry = YES;
        textField.delegate = self.passwordTwoObserver;
        [self.passwordOneObserver setNextTextField:textField withBlock:^(
            [twin tableView scrollToRowAtIndexPath:[NSIndexPath indexPathForItem:indexPath.row inSection:indexPath.section]
                atScrollPosition:UITableViewScrollPositionBottom
                animated:YES];
            );
    }
    }
}

} else if (indexPath.section == FKProfileSectionIndexGeneral) {
    if (indexPath.row == 0) {
        cell.textLabel.text = @"Gender";
        cell.detailTextLabel.text = self.profile.gender;
    } else if (indexPath.row == 1) {
        cell.textLabel.text = @"Birthday";
        cell.detailTextLabel.text = [NSDateFormatter localizedStringFromDate:self.profile.birthday
            dateStyle:NSDateFormatterMediumStyle
            timeStyle:NSDateFormatterNoStyle];
    }

    } else if (indexPath.row == 2) {
        cell.textLabel.text = @"Height";
        if (self.profile.height != nil) {
            NSInteger height = [self.profile.height integerValue];
            cell.detailTextLabel.text = [NSString stringWithFormat:@"%d' %d\"", height / 12, height % 12];
        } else {
            cell.detailTextLabel.text = nil;
        }
    }
}

} else if (indexPath.section == FKProfileSectionIndexGolf) {
    if (indexPath.row == 0) {
        cell.accessoryType = UITableViewCellAccessoryDisclosureIndicator;
        cell.textLabel.text = @"Handicap";
        cell.detailTextLabel.text = [self.profile.handicap description];
    } else if (indexPath.row == 1) {
        cell.accessoryType = UITableViewCellAccessoryDisclosureIndicator;
        cell.textLabel.text = @"Stance";
        cell.detailTextLabel.text = self.profile.stance;
    }
}

} else if (indexPath.section == FKProfileSectionIndexSave || indexPath.section == FKProfileSectionIndexDelete) {
    cell = [tableView dequeueReusableCellWithIdentifier:buttonCellIdentifier];
    NSUInteger labelTag = 1919;
    if (cell == nil) {
        cell = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleValue1 reuseIdentifier:buttonCellIdentifier];
        UIView *contentView = [cell contentView];
        UILabel *label = [[UILabel alloc] initWithFrame:[contentView frame]];
        [label setAutoresizingMask:UIViewAutoresizingFlexibleWidth];
        label.tag = labelTag;
        UIFont *boldFont = [UIFont boldSystemFontOfSize:[UIFont buttonFontSize]];
        [label setFont:boldFont];
        [label setTextAlignment:NSTextAlignmentCenter];
        [label setBackgroundColor:[UIColor clearColor]];
        [contentView addSubview:label];
        label.text = @"Save";
    }
}

if (indexPath.section == FKProfileSectionIndexSave) {
    cell.textLabel.textColor = [UIColor blackColor];
    if ([self.inRegistrationMode]) {
        UILabel *label = (UILabel *)[cell viewWithTag:labelTag];
        label.text = @"Join";
    }
}

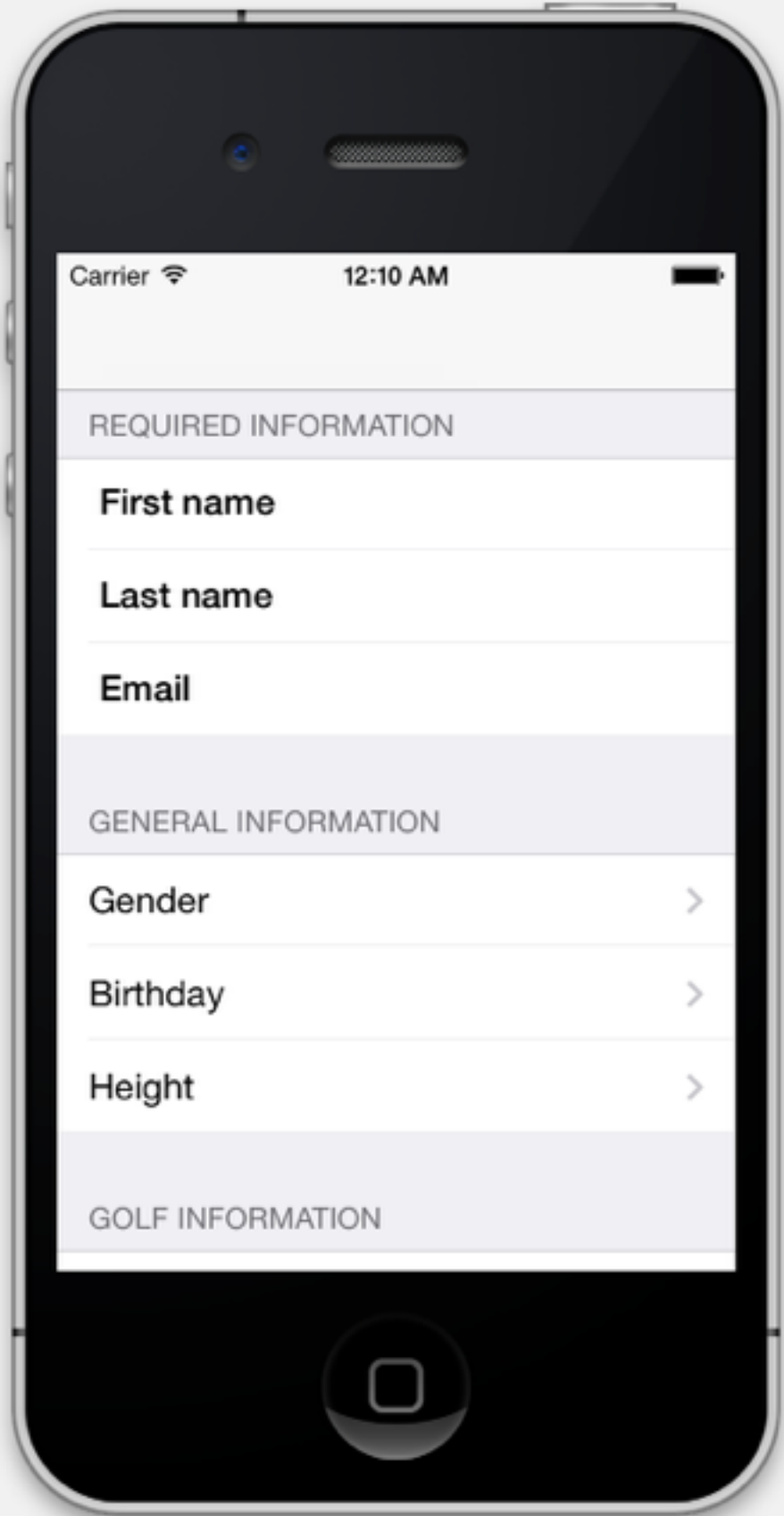
if ([self.profileInformationIsValid]) {
    cell.selectionStyle = UITableViewCellSelectionStyleGray;
    [[UILabel *)[cell viewWithTag:labelTag] setTextColor:[UIColor whiteColor]];
    cell.backgroundColor = [UIColor darkGrayColor];
}

} else {
    cell.selectionStyle = UITableViewCellStyleNone;
    cell.backgroundColor = [UIColor lightGrayColor];
    [[UILabel *)[cell viewWithTag:labelTag] setTextColor:[UIColor grayColor]];
}

} else if (indexPath.section == FKProfileSectionIndexDelete) {
    UILabel *label = (UILabel *)[cell viewWithTag:labelTag];
    label.text = @"Delete";
    cell.selectionStyle = UITableViewCellStyleNone;
    [[UILabel *)[cell viewWithTag:labelTag] setTextColor:[UIColor whiteColor]];
    cell.backgroundColor = [UIColor redColor];
}

}

return cell;
}
```



```

UITableViewCell *cell = nil;
if (indexPath.section == FKProfileSectionIndexGeneral || indexPath.section == FKProfileSectionIndexGolf) {
    cell = [tableView dequeueReusableCellWithIdentifier:@"ValueCell"];
    if (cell == nil) {
        cell = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleValue1 reuseIdentifier:@"ValueCell"];
        cell.accessoryType = UITableViewCellAccessoryDisclosureIndicator;
    }
}

if (indexPath.section == FKProfileSectionIndexRequired) {
    cell = [tableView dequeueReusableCellWithIdentifier:@"TextEntryCellIdentifier"];
    cell.selectionStyle = UITableViewCellStyleNone;

    UILabel *label = (UILabel *) [cell viewWithTag:10];
    UITextField *textField = (UITextField *) [cell viewWithTag:11];
    textField.secureTextEntry = NO;
    textField.keyboardType = UIKeyboardTypeDefault;

    UITableViewController *twinn = self;
    if (indexPath.row == 0) {
        label.text = @"First name";
        textField.text = self.profile.firstName;
        textField.delegate = self.firstNameObserver;
    } else if (indexPath.row == 1) {
        label.text = @"Last name";
        textField.text = self.profile.lastName;
        textField.delegate = self.lastNameObserver;
        [self.firstNameObserver setNextTextField:textField withBlock:^(
            [twinn.tableView scrollToRowAtIndex:indexPath.section:indexPath.row inSection:indexPath.section]
            atScrollPosition:UITableViewScrollPositionBottom
            animated:YES);
        ];
    }
} else if (indexPath.row == 2) {
    label.text = @"Email";
    textField.text = self.email;
    textField.delegate = self.emailObserver;
    textField.keyboardType = UIKeyboardTypeEmailAddress;
    [self.lastNameObserver setNextTextField:textField withBlock:^(
        [twinn.tableView scrollToRowAtIndex:indexPath.section:indexPath.row inSection:indexPath.section]
        atScrollPosition:UITableViewScrollPositionBottom
        animated:YES);
    ];
}
} else if (indexPath.row == 3) {
    label.text = @"Password";
    textField.text = self.passwordOne;
    textField.secureTextEntry = YES;
    textField.delegate = self.passwordOneObserver;
    [self.emailObserver setNextTextField:textField withBlock:^(
        [twinn.tableView scrollToRowAtIndex:indexPath.section:indexPath.row inSection:indexPath.section]
        atScrollPosition:UITableViewScrollPositionBottom
        animated:YES);
    ];
}
} else if (indexPath.row == 4) {
    label.text = @"Confirm";
    textField.text = self.passwordTwo;
    textField.secureTextEntry = YES;
    textField.delegate = self.passwordTwoObserver;
    [self.passwordOneObserver setNextTextField:textField withBlock:^(
        [twinn.tableView scrollToRowAtIndex:indexPath.section:indexPath.row inSection:indexPath.section]
        atScrollPosition:UITableViewScrollPositionBottom
        animated:YES);
    ];
}
}

if (indexPath.section == FKProfileSectionIndexGeneral) {
    if (indexPath.row == 0) {
        cell.textLabel.text = @"Gender";
        cell.detailTextLabel.text = self.profile.gender;
    } else if (indexPath.row == 1) {
        cell.textLabel.text = @"Birthday";
        cell.detailTextLabel.text = [NSDateFormatter localizedStringFromDate:self.profile.birthday
            dateStyle:NSDateFormatterMediumStyle
            timeStyle:NSDateFormatterNoStyle];
    }
} else if (indexPath.row == 2) {
    cell.textLabel.text = @"Height";
    if (self.profile.height != nil) {
        NSInteger height = [self.profile.height integerValue];
        cell.detailTextLabel.text = [NSString stringWithFormat:@"%d' %d\"", height / 12, height % 12];
    } else {
        cell.detailTextLabel.text = nil;
    }
}
}

if (indexPath.section == FKProfileSectionIndexGolf) {
    if (indexPath.row == 0) {
        cell.accessoryType = UITableViewCellAccessoryDisclosureIndicator;
        cell.textLabel.text = @"Handicap";
        cell.detailTextLabel.text = [self.profile.handicap description];
    } else if (indexPath.row == 1) {
        cell.accessoryType = UITableViewCellAccessoryDisclosureIndicator;
        cell.textLabel.text = @"Stance";
        cell.detailTextLabel.text = self.profile.stance;
    }
}

if (indexPath.section == FKProfileSectionIndexSave || indexPath.section == FKProfileSectionIndexDelete) {
    cell = [tableView dequeueReusableCellWithIdentifier:@"buttonCellIdentifier"];
    NSInteger labelTag = 1919;
    if (cell == nil) {
        cell = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleValue1 reuseIdentifier:@"buttonCellIdentifier"];
        UILabel *label = [UILabel alloc] initWithFrame:[contentView frame];
        UILabel *contentView = [UILabel alloc] initWithFrame:[contentView frame];
        label.tag = labelTag;
        UIFont *boldFont = [UIFont boldSystemFontOfSize:[UIFont buttonFontSize]];
        [label setFont:boldFont];
        [label setTextAlignment:NSTextAlignmentCenter];
        [label setBackgroundColor:[UIColor clearColor]];
        [contentView addSubview:label];
        label.text = @"Save";
    }
}

if (indexPath.section == FKProfileSectionIndexSave) {
    cell.textLabel.textColor = [UIColor blackColor];
    if ([self.inRegistrationMode]) {
        UILabel *label = (UILabel *) [cell viewWithTag:labelTag];
        label.text = @"Join";
    }
}

if ([self.profileInformationValid]) {
    cell.selectionStyle = UITableViewCellStyleGray;
    ([UILabel *) [cell viewWithTag:labelTag] setTextColor:[UIColor whiteColor]);
    cell.backgroundColor = [UIColor darkGrayColor];
} else {
    cell.selectionStyle = UITableViewCellStyleNone;
    cell.backgroundColor = [UIColor lightGrayColor];
    ([UILabel *) [cell viewWithTag:labelTag] setTextColor:[UIColor grayColor]);
}

if (indexPath.section == FKProfileSectionIndexDelete) {
    UILabel *label = (UILabel *) [cell viewWithTag:labelTag];
    label.text = @"Delete";
    cell.selectionStyle = UITableViewCellStyleNone;
    ([UILabel *) [cell viewWithTag:labelTag] setTextColor:[UIColor whiteColor]);
    cell.backgroundColor = [UIColor redColor];
}
}

return cell;
}

```

```

@interface RSMSPProfileViewController ()

@property (nonatomic, strong) RMSButtonCell *saveButton;
@property (nonatomic, strong) RMSFormSection *accountSection;
@property (nonatomic, strong) NSSpring *passwordOne;
@property (nonatomic, strong) NSSpring *passwordTwo;

@end

@implementation RSMSPProfileViewController

#pragma mark plist object mapping

- (NSDictionary *)objectSubstitutionDictionary {
    return @{
        @"self" : self,
        @"profile" : self.profile,
        @"selectedColor" : [UIColor whiteColor],
        @"blackColor" : [UIColor blackColor],
        @"redColor" : [UIColor redColor],
        @"selected" : self.isRegistrationMode ? @"on" : @"save",
        @"emailEnabled" : @(self.profile.isDefaultEmail),
        @"passwordRequired" : @(self.isRegistrationMode),
        @"selectedGender" : @(self.isRegistrationMode) ? self.profile.selectedGender : nil,
        @"gender" : @(self.profile.isMale ? @"male" : @"female")
    };
}

#pragma mark lifecycle

- (void)viewDidLoad {
    [super viewDidLoad];

    self.saveButton.enabled = [self.profileInformationValid];

    self.navigationController.rightBarButtonItems = [[UIBarButtonItem alloc] initWithTitle:@"Settings"
                                                style:UIBarButtonItemStyleBordered
                                                target:self
                                                action:[selector(settingsButtonAction)]];
}

- (void)viewDidLoad {
    [super viewDidLoad];

    [self startObserving];
}

- (voiddealloc {
    [self stopObserving];
}

- (voidstartObserving {
    for (NSString *key in [RMSProfile attributes]) {
        [self.profile addObserver:self forKeyPath:key
                        options:NSKeyValueObservingOptionInitial|NSKeyValueObservingOptionNew context:NULL];
    }
}

- (voidstopObserving {
    for (NSString *key in [RMSProfile attributes]) {
        [self.profile removeObserver:self forKeyPath:key];
    }
}

- (voidobserveValueForKeyPath:(NSString *)keyPath ofObject:(id)object change:(NSDictionary *)change context:(void *)context {
    self.title = self.profile.fullName;
    [self updateSaveButtonStatus];
}

NSLayoutConstraint *saveSectionIndex = [self.tableView indexForCell:self.saveButton] section;

[self.tableView reloadDataSections:[NSIndexSet indexSetWithIndex:saveSectionIndex]
        withRowAnimation:UITableViewRowAnimationNone];

- (voidupdateSaveButtonStatus {
    self.saveButton.enabled = [self.profileInformationValid];
}

- (voidupdatePasswordMessage {
    NSString *waitingFooter = self.accountSection.footerTitle + self.accountSection.footerTitle : @"";

    if ([self.passwordOne.length > 0] && [self.passwordTwo.length > 0]) {
        if ([self.passwordOne isEqualToString:self.passwordTwo]) {
            self.accountSection.footerTitle = @"Passwords don't match";
        } else if ([self.passwordOne.length < 6]) {
            self.accountSection.footerTitle = @"Passwords must be at least 6 characters long";
        } else {
            self.accountSection.footerTitle = @"";
        }
    } else {
        self.accountSection.footerTitle = @"";
    }

    if ([self.accountSection.footerTitle isEqualToString:self.waitingFooter]) {
        [self.tableView reloadDataSections:[NSIndexSet indexSetWithIndex:saveSectionIndex]
                withRowAnimation:UITableViewRowAnimationNone];
    }
}

- (voidpasswordOne:(NSString *)passwordOne {
    _passwordOne = passwordOne;
    [self updatePasswordMessage];
}

- (voidpasswordTwo:(NSString *)passwordTwo {
    _passwordTwo = passwordTwo;
    [self updatePasswordMessage];
}

#pragma mark actions

- (voidsettingsButtonAction:(id)sender {
    RMSSettingViewController *settingsViewController = [[RMSSettingViewController alloc] initWithStyle:UITableViewStyleGrouped description:@"NavigationController.settingsViewController.settingsViewController"];
    [self.navigationController presentViewController:navigationController animated:YES completion:nil];
}

- (voidsaveAction:(id)sender {
    if ([self.profileInformationValid]) {
        if ([self.isRegistrationMode]) {
            [self registerUser];
        } else {
            [self save];
        }
    }
}

- (voiddeleteAction:(id)sender {
    [self delete];
}

- (voidpushWordViewController {
    RMSWordViewController *wordViewController = [[RMSWordViewController alloc] initWithStyle:UITableViewStylePlain];
    [self.navigationController pushViewController:navigationController.navigationController animated:YES];
}

#pragma mark real work would occur here

- (voiddelete {
    [UIAlertView showAlertWebMessage:@"DELETE"];
}

- (voidsave {
    [UIAlertView showAlertWebMessage:@"SAVE"];
}

- (voidregisterUser {
    [UIAlertView showAlertWebMessage:@"REGISTER"];
}

#pragma mark validation, sort of

- (BOOL)profileInformationValid {
    BOOL profileInformationValid = [self.profile.isValid];
    if ([self.isRegistrationMode]) {
        profileInformationValid = profileInformationValid && [self.validateEmail:self.profile.email] && [self.validatePasswordPossiblyValid];
    }

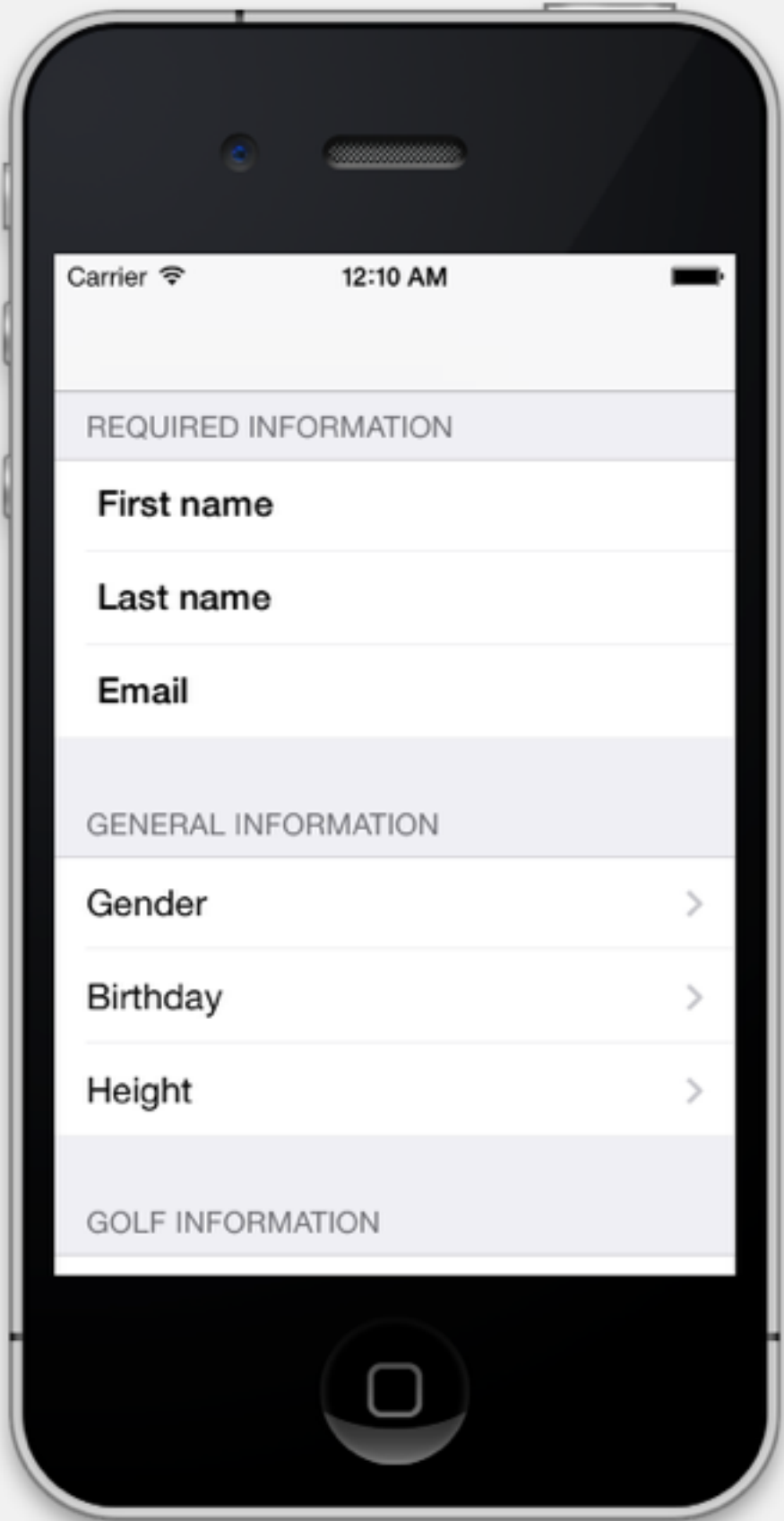
    return profileInformationValid;
}

- (BOOL)passwordsArePossiblyValid {
    return [self.passwordOne.length > 6] && [self.passwordOne isEqualToString:self.passwordTwo];
}

- (BOOL)validateEmail:(NSString *)candidate {
    NSString *emailRegex = @"^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$";
    NSPredicate *emailTest = [NSPredicate predicateWithFormat:@"%SELF MATCHES %@", emailRegex];
    return [emailTest evaluateWithObject:candidate];
}

@end

```



RoleModel

| |
|--------------------------------|
| RMSModel.h |
| RMSModel.m |
| RMSProfile.h |
| RMSProfile.m |
| RMSNamedColor.h |
| RMSNamedColor.m |
| Views |
| RMSBattingAveragePickerCell.h |
| RMSBattingAveragePickerCell.m |
| RMSHeightPickerCell.h |
| RMSHeightPickerCell.m |
| RMSNamedColorPickerCell.h |
| RMSNamedColorPickerCell.m |
| RMSWordCell.h |
| RMSWordCell.m |
| Controllers |
| Profile View Controller |
| RMSProfileViewController.h |
| RMSProfileViewController.m |
| ProfileViewController.plist |
| Settings View Controller |
| RMSSettingsViewController.h |
| RMSSettingsViewController.m |
| RMSSettingsV...ontroller.json |
| RMSSettingsV...ontroller.plist |
| Height Picker View Controller |
| Word View Controller |
| Resources |
| RMSHeightPickerView.storyboard |
| Images.xcassets |
| SDK Categories |
| UIAlertView+RMS.h |
| UIAlertView+RMS.m |
| Supporting Files |
| Table View Demo-Info.plist |
| InfoPlist.strings |
| main.m |
| Table View Demo-Prefix.pch |
| RMSTableViews |
| RMSTableViewController.h |
| RMSTableViewController.m |
| RMSTableViewSection.h |
| RMSTableViewSection.m |
| RMSTableViewCell.h |
| Dynamic Tables |
| RMSDynamicSection.h |
| RMSDynamicSection.m |
| RMSDynamicTableViewCell.h |

| Table View Demo | Table View Demo | Controllers | Profile |
|-------------------|-----------------|---------------------|---------|
| Key | Type | Value | |
| Root | Array | (6 items) | |
| Item 0 | Dictionary | (3 items) | |
| bindVariable | String | accountSection | |
| properties | Dictionary | (1 item) | |
| headerTitle | String | Account Information | |
| rows | Array | (5 items) | |
| Item 0 | Dictionary | (2 items) | |
| className | String | RMSTextEntryCell | |
| properties | Dictionary | (3 items) | |
| representedObject | String | :self | |
| keyPath | String | profile.firstName | |
| labelText | String | First name | |
| Item 1 | Dictionary | (2 items) | |
| className | String | RMSTextEntryCell | |
| properties | Dictionary | (3 items) | |
| Item 2 | Dictionary | (3 items) | |
| enabled | String | :emailEnabled | |
| className | String | RMSTextEntryCell | |
| properties | Dictionary | (3 items) | |
| Item 3 | Dictionary | (3 items) | |
| enabled | String | :passwordsEnabled | |
| className | String | RMSTextEntryCell | |
| properties | Dictionary | (4 items) | |
| Item 4 | Dictionary | (3 items) | |
| enabled | String | :passwordsEnabled | |
| className | String | RMSTextEntryCell | |
| properties | Dictionary | (4 items) | |
| representedObject | String | :self | |
| keyPath | String | passwordTwo | |
| labelText | String | Confirm | |
| secureTextEntry | String | 1 | |
| Item 1 | Dictionary | (2 items) | |
| properties | Dictionary | (1 item) | |
| headerTitle | String | General Information | |
| rows | Array | (2 items) | |
| Item 0 | Dictionary | (2 items) | |
| className | String | RMSArrayPickerCell | |
| properties | Dictionary | (4 items) | |
| representedObject | String | :profile | |
| keyPath | String | gender | |
| labelText | String | Gender | |
| choices | String | :genders | |
| Item 1 | Dictionary | (2 items) | |
| Item 2 | Dictionary | (2 items) | |
| Item 3 | Dictionary | (1 item) | |
| Item 4 | Dictionary | (2 items) | |

| |
|--------------------------------|
| RMSModel.h |
| RMSModel.m |
| RMSProfile.h |
| RMSProfile.m |
| RMSNamedColor.h |
| RMSNamedColor.m |
| Views |
| RMSBattingAveragePickerCell.h |
| RMSBattingAveragePickerCell.m |
| RMSHeightPickerCell.h |
| RMSHeightPickerCell.m |
| RMSNamedColorPickerCell.h |
| RMSNamedColorPickerCell.m |
| RMSWordCell.h |
| RMSWordCell.m |
| Controllers |
| Profile View Controller |
| RMSProfileViewController.h |
| RMSProfileViewController.m |
| ProfileViewController.plist |
| Settings View Controller |
| RMSSettingsViewController.h |
| RMSSettingsViewController.m |
| RMSSettingsV...ontroller.json |
| RMSSettingsV...ontroller.plist |
| Height Picker View Controller |
| Word View Controller |
| Resources |
| RMSHeightPickerView.storyboard |
| Images.xcassets |
| SDK Categories |
| UIAlertView+RMS.h |
| UIAlertView+RMS.m |
| Supporting Files |
| Table View Demo-Info.plist |
| InfoPlist.strings |
| main.m |
| Table View Demo-Prefix.pch |
| RMSTableViews |
| RMSTableViewController.h |
| RMSTableViewController.m |
| RMSTableViewSection.h |
| RMSTableViewSection.m |
| RMSTableViewCell.h |
| Dynamic Tables |
| RMSDynamicSection.h |
| RMSDynamicSection.m |
| RMSDynamicTableViewCell.h |

| Key | Type | Value |
|-------------------|------------|---------------------|
| Root | Array | (6 items) |
| Item 0 | Dictionary | (3 items) |
| bindVariable | String | accountSection |
| properties | Dictionary | (1 item) |
| headerTitle | String | Account Information |
| rows | Array | (5 items) |
| Item 0 | Dictionary | (2 items) |
| className | String | RMSTextEntryCell |
| properties | Dictionary | (3 items) |
| representedObject | String | :self |
| keyPath | String | profile.firstName |
| labelText | String | First name |
| Item 1 | Dictionary | (2 items) |
| className | String | RMSTextEntryCell |
| properties | Dictionary | (3 items) |
| Item 2 | Dictionary | (3 items) |
| enabled | String | :emailEnabled |
| className | String | RMSTextEntryCell |
| properties | Dictionary | (3 items) |
| Item 3 | Dictionary | (3 items) |
| enabled | String | :passwordsEnabled |
| className | String | RMSTextEntryCell |
| properties | Dictionary | (4 items) |
| Item 4 | Dictionary | (3 items) |
| enabled | String | :passwordsEnabled |
| className | String | RMSTextEntryCell |
| properties | Dictionary | (4 items) |
| representedObject | String | :self |
| keyPath | String | passwordTwo |
| labelText | String | Confirm |
| secureTextEntry | String | 1 |
| Item 1 | Dictionary | (2 items) |
| properties | Dictionary | (1 item) |
| headerTitle | String | General Information |
| rows | Array | (2 items) |
| Item 0 | Dictionary | (2 items) |
| className | String | RMSArrayPickerCell |
| properties | Dictionary | (4 items) |
| representedObject | String | :profile |
| keyPath | String | gender |
| labelText | String | Gender |
| choices | String | :genders |
| Item 1 | Dictionary | (2 items) |
| Item 2 | Dictionary | (2 items) |
| Item 3 | Dictionary | (1 item) |
| Item 4 | Dictionary | (2 items) |

Top level items describe Sections

Rows in
Sections
describe Cells

Top level
items describe
Sections

| Key | Type | Value |
|-------------------|------------|---------------------|
| Root | Array | (6 items) |
| Item 0 | Dictionary | (3 items) |
| bindVariable | String | accountSection |
| properties | Dictionary | (1 item) |
| headerTitle | String | Account Information |
| rows | Array | (5 items) |
| Item 0 | Dictionary | (2 items) |
| className | String | RMSTextFieldCell |
| properties | Dictionary | (3 items) |
| representedObject | String | :self |
| keyPath | String | profile.firstName |
| labelText | String | First name |
| Item 1 | Dictionary | (2 items) |
| className | String | RMSTextFieldCell |
| properties | Dictionary | (3 items) |
| Item 2 | Dictionary | (3 items) |
| enabled | String | :emailEnabled |
| className | String | RMSTextFieldCell |
| properties | Dictionary | (3 items) |
| Item 3 | Dictionary | (3 items) |
| enabled | String | :passwordsEnabled |
| className | String | RMSTextFieldCell |
| properties | Dictionary | (4 items) |
| Item 4 | Dictionary | (3 items) |
| enabled | String | :passwordsEnabled |
| className | String | RMSTextFieldCell |
| properties | Dictionary | (4 items) |
| representedObject | String | :self |
| keyPath | String | passwordTwo |
| labelText | String | Confirm |
| secureTextEntry | String | 1 |
| Item 1 | Dictionary | (2 items) |
| properties | Dictionary | (1 item) |
| headerTitle | String | General Information |
| rows | Array | (2 items) |
| Item 0 | Dictionary | (2 items) |
| className | String | RMSArrayPickerCell |
| properties | Dictionary | (4 items) |
| representedObject | String | :profile |
| keyPath | String | gender |
| labelText | String | Gender |
| choices | String | :genders |
| Item 1 | Dictionary | (2 items) |
| Item 2 | Dictionary | (2 items) |
| Item 3 | Dictionary | (1 item) |
| Item 4 | Dictionary | (2 items) |

Rows in
Sections
describe Cells

Various
Generic Form
Cells already
available

Top level
items describe
Sections

| Key | Type | Value |
|-------------------|------------|---------------------|
| Root | Array | (6 items) |
| Item 0 | Dictionary | (3 items) |
| bindVariable | String | accountSection |
| properties | Dictionary | (1 item) |
| headerTitle | String | Account Information |
| rows | Array | (5 items) |
| Item 0 | Dictionary | (2 items) |
| className | String | RMSTextFieldCell |
| properties | Dictionary | (3 items) |
| representedObject | String | :self |
| keyPath | String | profile.firstName |
| labelText | String | First name |
| Item 1 | Dictionary | (2 items) |
| className | String | RMSTextFieldCell |
| properties | Dictionary | (3 items) |
| Item 2 | Dictionary | (3 items) |
| enabled | String | :emailEnabled |
| className | String | RMSTextFieldCell |
| properties | Dictionary | (3 items) |
| Item 3 | Dictionary | (3 items) |
| enabled | String | :passwordsEnabled |
| className | String | RMSTextFieldCell |
| properties | Dictionary | (4 items) |
| Item 4 | Dictionary | (3 items) |
| enabled | String | :passwordsEnabled |
| className | String | RMSTextFieldCell |
| properties | Dictionary | (4 items) |
| representedObject | String | :self |
| keyPath | String | passwordTwo |
| labelText | String | Confirm |
| secureTextEntry | String | 1 |
| Item 1 | Dictionary | (2 items) |
| properties | Dictionary | (1 item) |
| headerTitle | String | General Information |
| rows | Array | (2 items) |
| Item 0 | Dictionary | (2 items) |
| className | String | RMSArrayPickerCell |
| properties | Dictionary | (4 items) |
| representedObject | String | :profile |
| keyPath | String | gender |
| labelText | String | Gender |
| choices | String | :genders |
| Item 1 | Dictionary | (2 items) |
| Item 2 | Dictionary | (2 items) |
| Item 3 | Dictionary | (1 item) |
| Item 4 | Dictionary | (2 items) |

Rows in
Sections
describe Cells

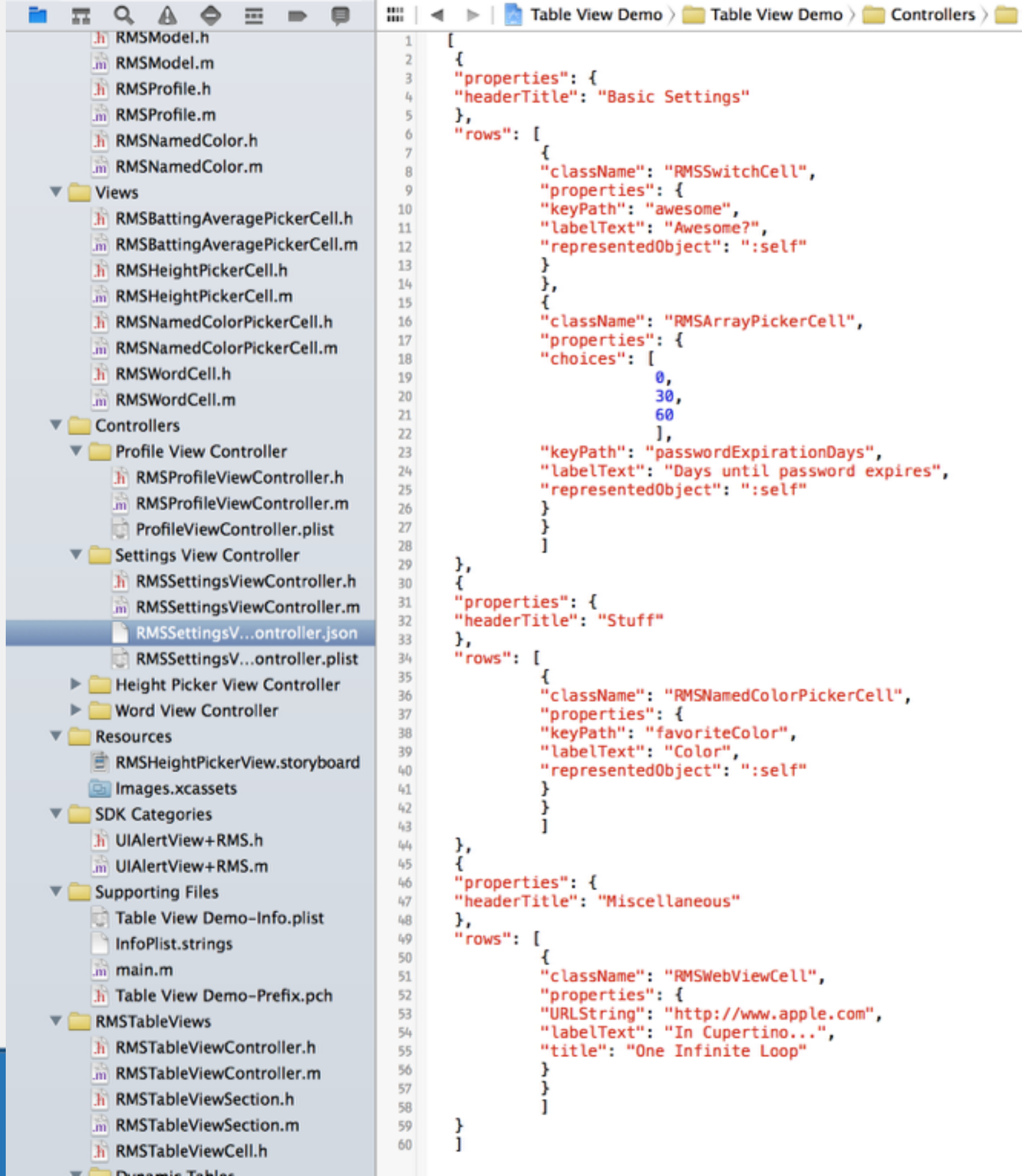
Various
Generic Form
Cells already
available

Top level
items describe
Sections

Things that
don't go well in Plists
can identify key to have
controller interpret &
substitute

| Key | Type | Value |
|-------------------|------------|---------------------|
| Root | Array | (6 items) |
| Item 0 | Dictionary | (3 items) |
| bindVariable | String | accountSection |
| properties | Dictionary | (1 item) |
| headerTitle | String | Account Information |
| rows | Array | (5 items) |
| Item 0 | Dictionary | (2 items) |
| className | String | RMSTextFieldCell |
| properties | Dictionary | (3 items) |
| representedObject | String | :self |
| keyPath | String | profile.firstName |
| labelText | String | First name |
| Item 1 | Dictionary | (2 items) |
| className | String | RMSTextFieldCell |
| properties | Dictionary | (3 items) |
| Item 2 | Dictionary | (3 items) |
| enabled | String | :emailEnabled |
| className | String | RMSTextFieldCell |
| properties | Dictionary | (3 items) |
| Item 3 | Dictionary | (3 items) |
| enabled | String | :passwordsEnabled |
| className | String | RMSTextFieldCell |
| properties | Dictionary | (4 items) |
| Item 4 | Dictionary | (3 items) |
| enabled | String | |
| className | String | |
| properties | Dictionary | (1 item) |
| representedObject | String | |
| keyPath | String | |
| labelText | String | |
| secureTextEntry | String | |
| Item 1 | Dictionary | (1 item) |
| properties | Dictionary | (1 item) |
| headerTitle | String | |
| rows | Array | (1 item) |
| Item 0 | Dictionary | (1 item) |
| className | String | |
| properties | Dictionary | (1 item) |
| representedObject | String | :profile |
| keyPath | String | gender |
| labelText | String | Gender |
| choices | String | :genders |
| Item 1 | Dictionary | (2 items) |
| Item 2 | Dictionary | (2 items) |
| Item 3 | Dictionary | (1 item) |
| Item 4 | Dictionary | (2 items) |

RoleModel



Can use
JSON if you
don't like Plists

The screenshot shows an Xcode project named "Table View Demo". The left sidebar displays the project's file structure, including source files like `RMSModel.h`, `RMSModel.m`, `RMSProfile.h`, `RMSProfile.m`, `RMSNamedColor.h`, and `RMSNamedColor.m`. It also shows a "Views" folder with various picker cells and a "Resources" folder with a storyboard and image assets. The right pane shows the content of the selected file, `RMSSettingsViewController.json`, which is a JSON configuration for a settings view controller. The JSON defines three sections: "Basic Settings", "Stuff", and "Miscellaneous", each with a list of rows and their corresponding cell classes and properties.

```
1  [
2  {
3    "properties": {
4      "headerTitle": "Basic Settings"
5    },
6    "rows": [
7      {
8        "className": "RMSSwitchCell",
9        "properties": {
10         "keyPath": "awesome",
11         "labelText": "Awesome?",
12         "representedObject": ":self"
13       }
14     },
15     {
16       "className": "RMSArrayPickerCell",
17       "properties": {
18         "choices": [
19           0,
20           30,
21           60
22         ],
23         "keyPath": "passwordExpirationDays",
24         "labelText": "Days until password expires",
25         "representedObject": ":self"
26       }
27     }
28   ],
29   {
30     "properties": {
31       "headerTitle": "Stuff"
32     },
33     "rows": [
34       {
35         "className": "RMSNamedColorPickerCell",
36         "properties": {
37           "keyPath": "favoriteColor",
38           "labelText": "Color",
39           "representedObject": ":self"
40         }
41       }
42     ]
43   },
44   {
45     "properties": {
46       "headerTitle": "Miscellaneous"
47     },
48     "rows": [
49       {
50         "className": "RMSWebViewController",
51         "properties": {
52           "URLString": "http://www.apple.com",
53           "labelText": "In Cupertino...",
54           "title": "One Infinite Loop"
55         }
56       }
57     ]
58   }
59 ]
60 ]
```

The form descriptor drives the construction of the table view

```
- (NSArray *)generateSections {  
    NSMutableArray *sections = [NSMutableArray array];  
  
    for (NSDictionary *rawSection in self.descriptor) {  
        /* Sections and Cells are built here */  
    }  
  
    return sections;  
}
```

Form View Controller Instantiation

- (**id**)initWithStyle:(UITableViewControllerStyle)style
descriptor:(NSArray *)descriptor;
- (**id**)initWithStyle:(UITableViewControllerStyle)style
descriptorNamed:(NSString *)descriptorName;
- (**id**)initWithStyle:(UITableViewControllerStyle)style
rawDescriptor:(NSData *)rawDescriptor
type:(RMSFormDescriptorType)descriptorType;

Form View Controller Instantiation

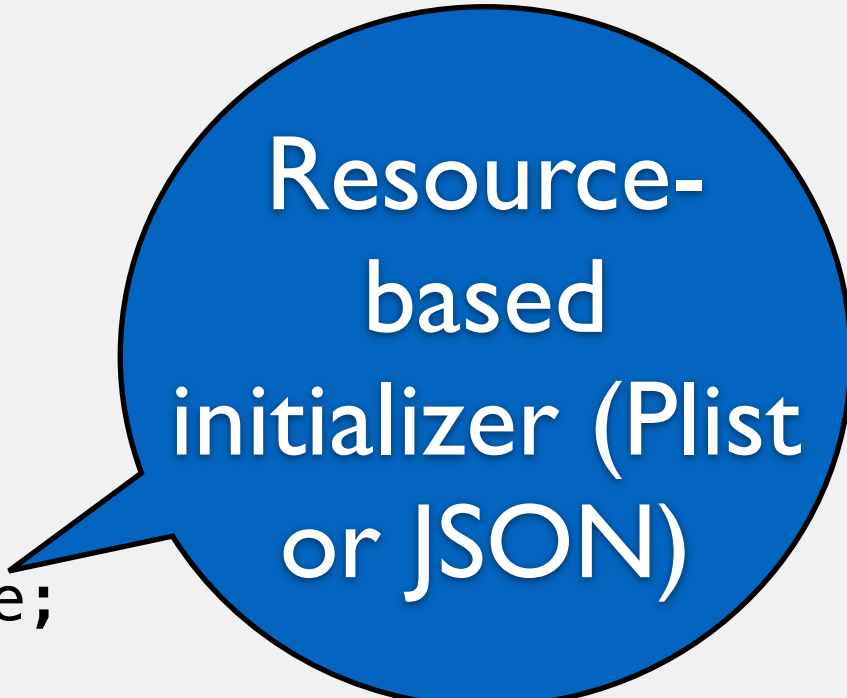


Designated
Initializer

- (**id**) initWithStyle:(UITableViewStyle)style
descriptor:(NSArray *)descriptor;
- (**id**) initWithStyle:(UITableViewStyle)style
descriptorNamed:(NSString *)descriptorName;
- (**id**) initWithStyle:(UITableViewStyle)style
rawDescriptor:(NSData *)rawDescriptor
type:(RMSFormDescriptorType)descriptorType;

Form View Controller Instantiation

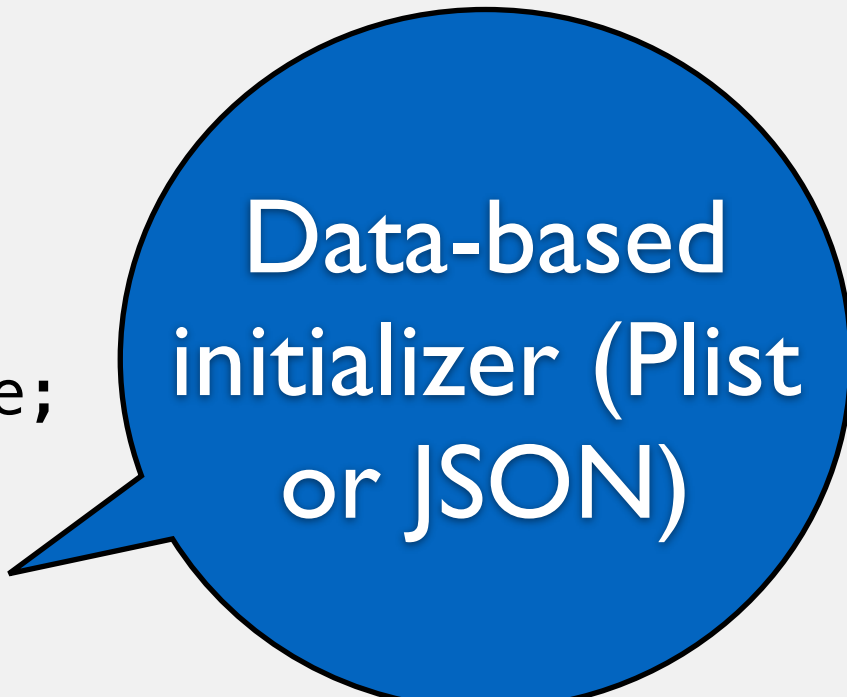
- (**id**)initWithStyle:(UITableViewStyle)style
descriptor:(NSArray *)descriptor;
- (**id**)initWithStyle:(UITableViewStyle)style
descriptorNamed:(NSString *)descriptorName;
- (**id**)initWithStyle:(UITableViewStyle)style
rawDescriptor:(NSData *)rawDescriptor
type:(RMSFormDescriptorType)descriptorType;



Resource-
based
initializer (Plist
or JSON)

Form View Controller Instantiation

- (**id**)initWithStyle:(UITableViewStyle)style
descriptor:(NSArray *)descriptor;
- (**id**)initWithStyle:(UITableViewStyle)style
descriptorNamed:(NSString *)descriptorName;
- (**id**)initWithStyle:(UITableViewStyle)style
rawDescriptor:(NSData *)rawDescriptor
type:(RMSFormDescriptorType)descriptorType;



Data-based
initializer (Plist
or JSON)

Demo

Building a Simple Form

Demo

Building a Simple Form

- Section definition

Demo

Building a Simple Form

- Section definition
- Data binding

Demo

Building a Simple Form

- Section definition
- Data binding
- Object substitution

Demo

Building a Simple Form


- Section definition
- Data binding
- Object substitution
- Cell properties

Form View Controller Instantiation

- (**id**)initWithStyle:(UITableViewControllerStyle)style
descriptor:(NSArray *)descriptor;
- (**id**)initWithStyle:(UITableViewControllerStyle)style
descriptorNamed:(NSString *)descriptorName;
- (**id**)initWithStyle:(UITableViewControllerStyle)style
rawDescriptor:(NSData *)rawDescriptor
type:(RMSFormDescriptorType)descriptorType;

Form View Controller Instantiation

- (**id**)initWithStyle:(UITableViewControllerStyle)style
descriptor:(NSArray *)descriptor;
- (**id**)initWithStyle:(UITableViewControllerStyle)style
descriptorNamed:(NSString *)descriptorName;
- (**id**)initWithStyle:(UITableViewControllerStyle)style
rawDescriptor:(NSData *)rawDescriptor
type:(RMSFormDescriptorType)descriptorType;



Supports
externalizing
the form
descriptor

Demo

Demo

- Resource-based Plist

Demo

- Resource-based Plist
- Resource-based JSON

Demo

- Resource-based Plist
- Resource-based JSON
- Web-based JSON

Demo

- Resource-based Plist
- Resource-based JSON
- Web-based JSON



Summary

Summary

- TableViews often require much tedium when not in the simplest form

Summary

- TableViews often require much tedium when not in the simplest form
 - Code becomes hard to parse

Summary

- TableViews often require much tedium when not in the simplest form
 - Code becomes hard to parse
 - Few interesting things happening (mostly tying objects to the right cells)

Summary

- TableViews often require much tedium when not in the simplest form
 - Code becomes hard to parse
 - Few interesting things happening (mostly tying objects to the right cells)
- Sections are objects that are implied but missing from the Object Model

Summary

- TableViews often require much tedium when not in the simplest form
 - Code becomes hard to parse
 - Few interesting things happening (mostly tying objects to the right cells)
- Sections are objects that are implied but missing from the Object Model
- Make Cells more intelligent about the objects they represent

UITableView Pain Reliever

<https://github.com/RoleModel/RMSTableViews>

Available as a CocoaPod, docs on cocoapods.org

Ken Auer

ken.auer@rolemodelsoftware.com

@kauerrolemodel

Tony Ingraldi

tony.ingraldi@rolemodelsoftware.com