

## Overview

- Part 1: Introduction to UITableView
- Part 2: Introduction to Fetching Content from the Web
- Part 3: Flickr TableView

## Submission

Lab evaluation form

Online Lab feedback form

Note: If you are unable to complete the lab during this week's lab period, please be ready to demonstrate before the beginning of the next lab.

## Part 1: Introduction to UITableView

UITableView is a great way to organize lists of content. A typical UITableView consists of one or more sections, which consist of UITableViewCell cells (a cell is a row in the table).

**Step 1)** Create a new project called **Table** from the Single View application template.

**Step 2)** Open the view controller's .xib file or the .storyboard.

In Interface Builder:

- 1) Add a UITableViewController onto the scene.
- 2) Edit the class of the UITableViewController to use your own custom view controller class
- 3) Ensure that your view controller is both the delegate and datasource of the UITableView. You can check by right-clicking on the Table View.
- 4) Save the .xib or storyboard.

You have now made your view controller the delegate and dataSource for the table view.

**Step 3)** In Xcode, declare in the header file that your view controller implements the delegate protocol for UITableViewDelegate and UITableViewDataSource:

```
@interface TableViewController : UIViewController <UITableViewDelegate, UITableViewDataSource>
```

Another way to declare this is to change the superclass you inherit to UITableViewController. UITableViewController is a wrapper on the UIViewController class that includes the two delegate protocols. If you use this method, make sure the view property of your TableViewController is set to the UITableView. Do this by dragging a connection from File's Owner / View Controller -> view to the TableView in Interface Builder.

```
@interface TableViewController : UITableViewController
```

**Step 4)** Implement methods from the UITableViewDelegate and UITableViewDataSource protocols. Most methods of the protocols have a default implementation. Some interesting methods for you to consider are:

```
numberOfSectionsInTableView:  
tableView:numberOfRowsInSection:  
tableView:titleForHeaderInSection:  
tableView:didSelectRowAtIndexPath:
```

Most of these methods are very simple to implement, requiring you to return a number or an NSString \*. The most complicated to implement is tableView:cellForRowAtIndexPath:, but a sample implementation has been provided below:

```
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath {
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:@"MainCell"];
    if (cell == nil) {
        cell = [[UITableViewCell alloc] initWithFrame:CGRectZero reuseIdentifier:@"MainCell"];
        [cell autorelease];
    }
    cell.textLabel.text = @"My Cell";
    //set other properties of cell

    return cell;
}
```

Notice how memory is saved by reusing the same instance of UITableViewCell. In cases where most cells in your table are similar, you'll want to follow the technique of the example. The caller will call the tableView:cellForRowAtIndexPath: method for every visible cell when it needs to draw that cell. For the first call, we allocate memory for a new UITableViewCell and set its text. After we pass back the cell to the caller, it will render the cell. For subsequent calls, we reuse the same cell (and reuse the same memory) and simply change the text.

**Step 5)** Implement a simple application that contains **at least 2 sections and at least 4 rows** in a table view. You may use the following example:

Section 0: "Countries to Visit"

Row 0: "England"

Row 1: "France"

Row 2: "Italy"

Section 1: "American Destinations"

Row 0: "Hawaii"

Row 1: "Disney World"

**Step 6)** You can wait to demo your code until you are finished with Part 3. The sections of the lab build upon each other.

## Part 2: Introduction to Fetching Content from the Web

In this section, you will use Flickr's API to fetch an image from a web-request and display the image on in the simulator.

<http://www.flickr.com/services/api/>

**Step 1)** Reuse your project from Part 1.

**Step 2)** Download JSON.zip from the course webpage.

Note: JSON (JavaScript Object Notation) is a widely used standard for serializing objects used in web development.

**Step 3)** Add the JSON classes to your project.

**Step 4)** Import JSON.h in your ViewController's header files.

**Step 5)** Review the following code and answer the questions in the Evaluation Form. Copy the code and use the debugger to fully understand what is happening.

```
NSMutableArray *photoNames = [[NSMutableArray alloc] init];
NSMutableArray *photoURLs = [[NSMutableArray alloc] init];

NSString *flickrAPIKey = @"78bd5fd661d75a1703ae1c533a30fae1";
NSString *apiCall = @"http://api.flickr.com/services/rest/?method=flickr.photos.search&api_key=%@&tags=%@&per_page=20&format=json&nojsoncallback=1";
NSString *urlString = [NSString stringWithFormat:apiCall, flickrAPIKey, @"iastate"];
NSURL *url = [NSURL URLWithString:urlString];

NSString *jsonString = [NSString stringWithContentsOfURL:url encoding:NSUTF8StringEncoding error:nil];
NSDictionary *results = [jsonString JSONValue];

NSArray *photos = [[results objectForKey:@"photos"] objectForKey:@"photo"];
for (NSDictionary *photo in photos) {
    // Get the title for each photo
    NSString *title = [photo objectForKey:@"title"];
    [photoNames addObject:(title.length > 0 ? title : @"Untitled")];

    // Construct the URL for each photo
    NSString *photoURLString = [NSString stringWithFormat:@"http://farm%@.static.flickr.com/%@/%@_%@_s.jpg",
        [photo objectForKey:@"farm"],
        [photo objectForKey:@"server"],
        [photo objectForKey:@"id"],
        [photo objectForKey:@"secret"]];
    [photoURLs addObject:[NSURL URLWithString:photoURLString]];
}
```

Note: You may want to type the URL into a web browser to see what is returned. Using GET variables (or POST variables) to pass a server-side script information is a common practice in web development.

[http://api.flickr.com/services/rest/?method=flickr.photos.search&api\\_key=78bd5fd661d75a1703ae1c533a30fae1&tags=iastate&per\\_page=10&format=json&nojsoncallback=1](http://api.flickr.com/services/rest/?method=flickr.photos.search&api_key=78bd5fd661d75a1703ae1c533a30fae1&tags=iastate&per_page=10&format=json&nojsoncallback=1)

Information about GET variables:

<http://www.myurl.com/test.php?var1=value1&var2=value2&var3=value3>

A ? mark delimits the URL from the GET variables. The variables themselves are delimited by &. Each variable has a name and a value, separated by an =. In a simple PHP script, the code can access these variables through the globally defined `_GET` dictionary. For example, if test.php were:

```
<? echo _GET['var1']; ?>
```

Then going to the sample URL would return a webpage that contains one word: value1.

### Part 3: Flickr TableView

Using the provided code from Part 2, you will update your TableView from Part 1 to display photos retrieved from your Flickr API fetch.

**Step 1)** Answer the questions for Part 3 on the Evaluation Form. You can explore the documentation by Ctrl-clicking (right clicking) on keywords in Xcode or by going to [developer.apple.com](http://developer.apple.com).

**Step 2)** Alter your code from Part 1 to display the photos returned from the web request.

The following convenience methods of the NSData and UIImage classes will be helpful:

+ (NSData \*) dataWithContentsOfURL:(NSURL \*)url;

+ (UIImage \*) imageWithData:(NSData \*)data;

**Step 4)** When your app has images displayed in each row of the TableView, demo your app to the TA.