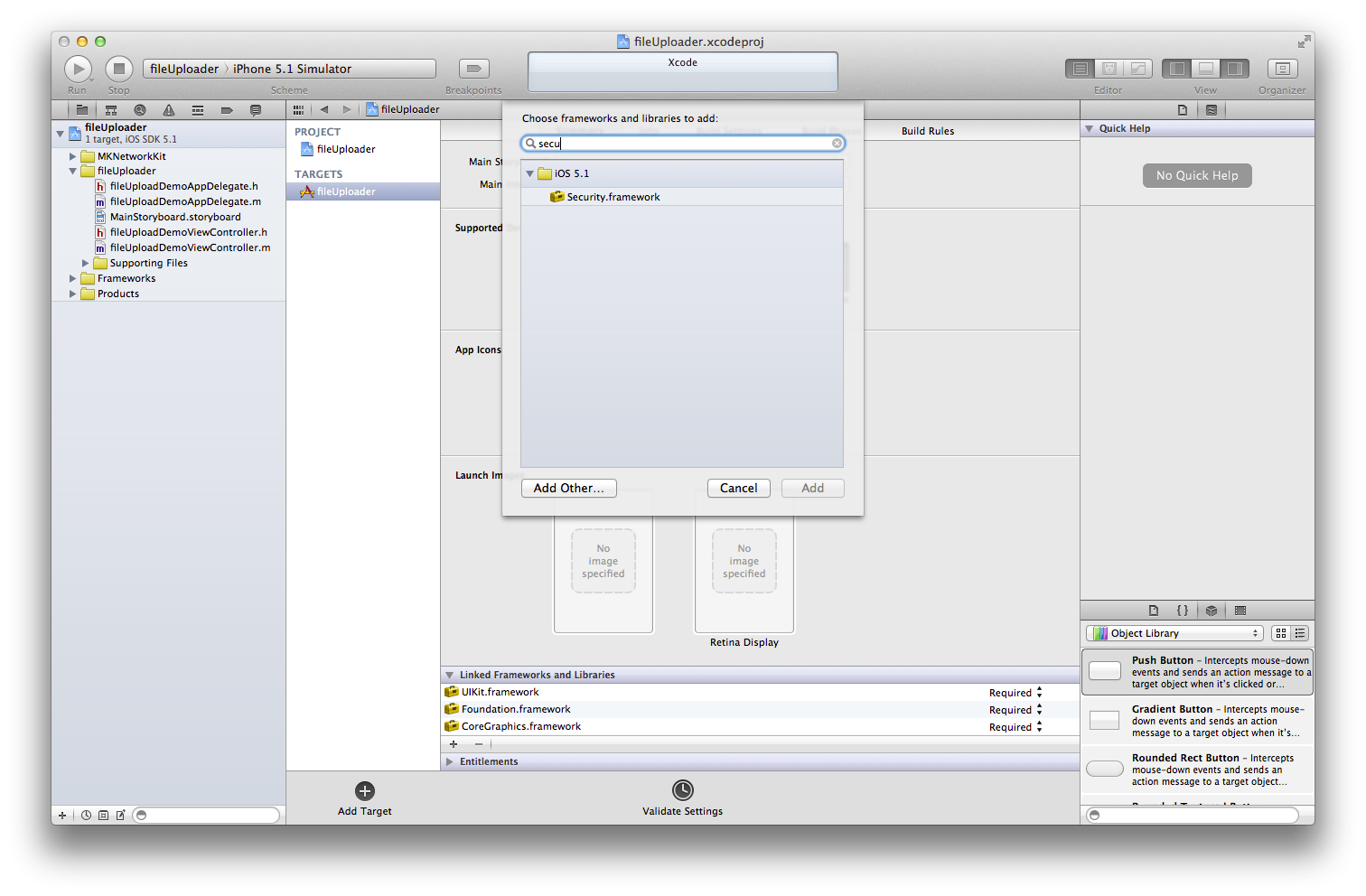
**How to Upload Pictures to a Server**

**Step 1: INSTALL MKNETWORKKIT- This is an engine that controls uploads…especially large ones by packeting the data**

**Clone the latest version of MKNetworkKit over at github (**[**https://github.com/MugunthKumar/MKNetworkKit**](https://github.com/MugunthKumar/MKNetworkKit)**).**

**It is important to grab the latest version as there was a small bug with POST. Drag the MKNetworkKit directory into your project within Xcode. Make sure to copy the items over if needed. You can watch me do this on the screen.**

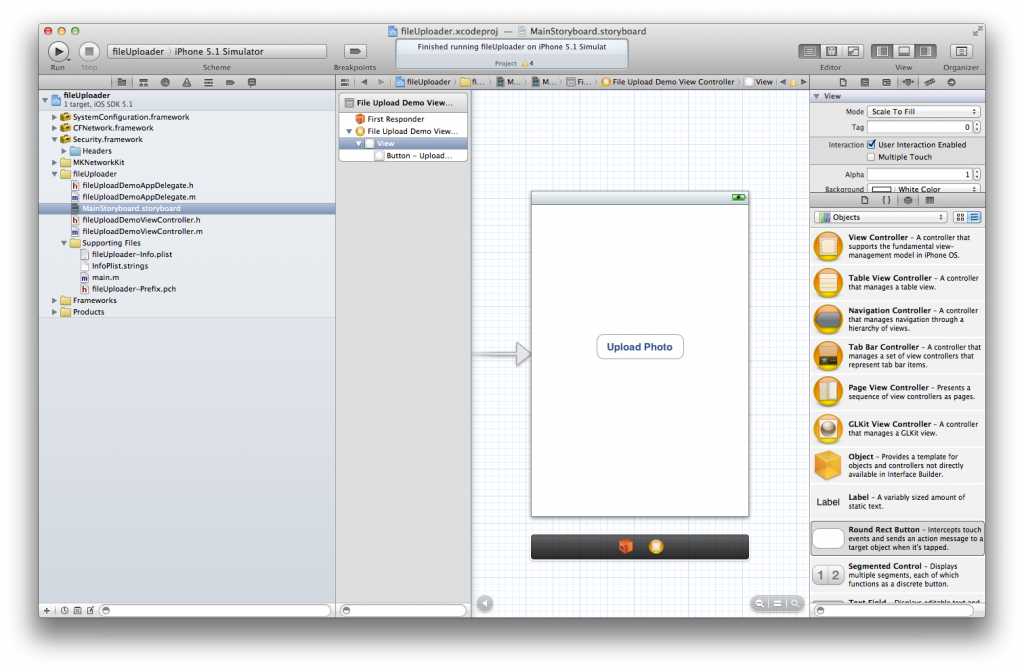
**Next, add the following built-in frameworks to your target: CFNetwork.Framework, SystemConfiguration.framework and Security.framework. You can do this in the Build Phases area under the Link Binary section (We’ve done this before).**

**Add #import “MKNetworkKit.h” within the PCH file in the supporting files folder. Be careful of the double quotes! Remember last week**

**MKNetworkKit has support for both iOS and Mac builds. Since we’re building an iOS app, remove the NSAlert+MKNetworkKitAdditions.h and NSAlert+MKNetworkKitAdditions.m files from the Categories folder. The MKNetworkKit should be installed! Do a build to ensure you’re good to go. You’ll get some dev triggered warnings and that’s ok. Carry on.**

**Step 2: SELECT THE FILE TO UPLOAD**

**For the sake of simplicity, let’s upload a photo. First, add an UIButton to the view and give it a title, “Upload Photo”.**

****

**Add a connection to the h file. Make it an action named uploadPhoto**

**Give the user some options when the button is clicked.**

**To accomplish this, use an UIActionSheet. Be sure to add UIActionSheetDelegate to the header file (See directly below).**

**Within the UIActionSheet’s delegate method, we need to determine which option was selected on the UIActionSheet.**

**Your ViewController.h file (probably named fileUploadViewController.h) should look like this:**

#import <UIKit/UIKit.h>

#import "fileUploadEngine.h"

@interface fileUploadDemoViewController : UIViewController <UIActionSheetDelegate, UIImagePickerControllerDelegate, UINavigationControllerDelegate>

@property (strong, nonatomic) fileUploadEngine \*flUploadEngine;

@property (strong, nonatomic) MKNetworkOperation \*flOperation;

@end

**This is that action sheet method…to be added to the fileUploadViewController.m**

- (void)actionSheet:(UIActionSheet \*)modalView clickedButtonAtIndex:(NSInteger)buttonIndex

{

switch (buttonIndex)

{

case 0:

{

if ([UIImagePickerController isSourceTypeAvailable:UIImagePickerControllerSourceTypeCamera]) {

UIImagePickerController \*imagePicker = [[UIImagePickerController alloc] init];

imagePicker.sourceType = UIImagePickerControllerSourceTypeCamera;

imagePicker.delegate = self;

imagePicker.cameraFlashMode = UIImagePickerControllerCameraFlashModeOff;

imagePicker.allowsEditing = NO;

[self presentModalViewController:imagePicker animated:YES];

}

else {

UIAlertView \*alert;

alert = [[UIAlertView alloc] initWithTitle:@"Error"

message:@"This device doesn't have a camera."

delegate:self cancelButtonTitle:@"Ok"

otherButtonTitles:nil];

[alert show];

}

break;

}

case 1:

{

if ([UIImagePickerController isSourceTypeAvailable:UIImagePickerControllerSourceTypePhotoLibrary]) {

UIImagePickerController \*imagePicker = [[UIImagePickerController alloc] init];

imagePicker.sourceType = UIImagePickerControllerSourceTypePhotoLibrary;

imagePicker.delegate = self;

imagePicker.allowsEditing = NO;

[self presentModalViewController:imagePicker animated:YES];

}

else {

UIAlertView \*alert;

alert = [[UIAlertView alloc] initWithTitle:@"Error"

message:@"This device doesn't support photo libraries."

delegate:self cancelButtonTitle:@"Ok"

otherButtonTitles:nil];

[alert show];

}

break;

}

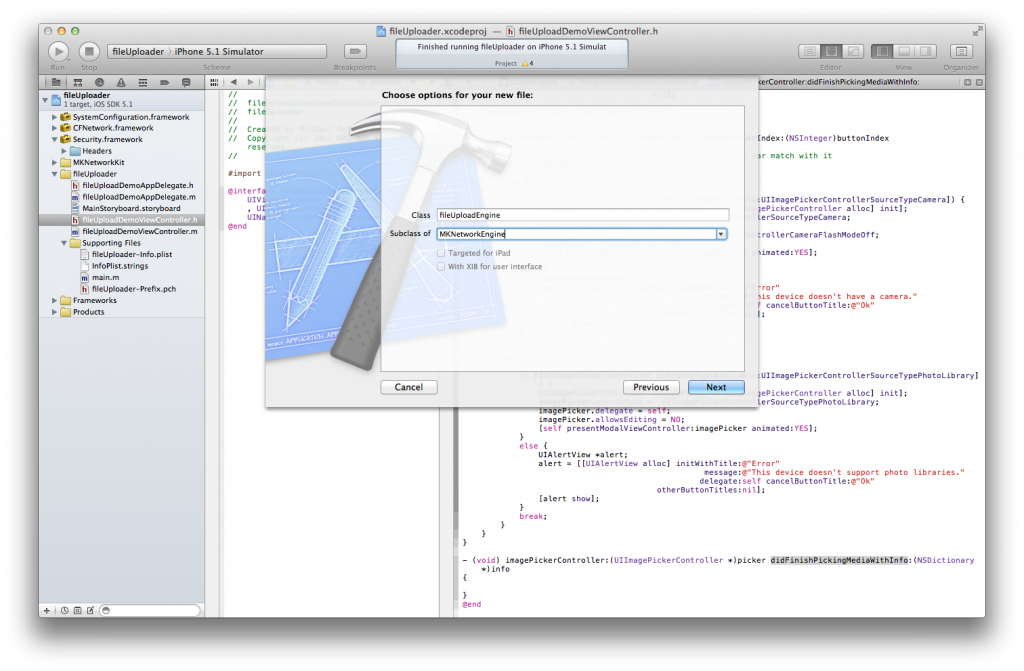
}

}

**Step 3: CREATE A NETWORK ENGINE**

**The MKNetworkEngine object manages the queues, caching, and other connectivity goodness that we, for the most part, don’t have to worry about.**

**To create one, add a new objective C class named fileUploadEngine with the subclass MKNetworkEngine. Our engine will just have one defined method in it, postDataToServer.**



**fileUploadEngine.h will look like this:**

#import "MKNetworkEngine.h"

@interface fileUploadEngine : MKNetworkEngine

-(MKNetworkOperation \*) postDataToServer:(NSMutableDictionary \*)params path:(NSString \*)path;

@end

**fileUploadEngine.m will look like this:**

#import "fileUploadEngine.h"

@implementation fileUploadEngine

-(MKNetworkOperation \*) postDataToServer:(NSMutableDictionary \*)params path:(NSString \*)path {

MKNetworkOperation \*op = [self operationWithPath:path

params:params

httpMethod:@"POST"

ssl:NO];

return op;

}

@end

**Time to add the UIImage Picker method:**

- (void) imagePickerController:(UIImagePickerController \*)picker didFinishPickingMediaWithInfo:(NSDictionary \*)info

{

[self dismissModalViewControllerAnimated:YES];

NSData \*image = UIImageJPEGRepresentation([info objectForKey:UIImagePickerControllerOriginalImage], 0.1);

self.flUploadEngine = [[fileUploadEngine alloc] initWithHostName:@"mm214.com" customHeaderFields:nil];

**//the above is our host…don’t worry, I have a folder for all of you**

NSMutableDictionary \*postParams = [NSMutableDictionary dictionaryWithObjectsAndKeys:

@"testApp", @"appID",

nil];

self.flOperation = [self.flUploadEngine postDataToServer:postParams path:@"/test/students/yourfirstinitiallastname/post.php"];

**//so Tamara Harding would be tharding (/test/students/tharding/post.php)**

[self.flOperation addData:image forKey:@"userfl" mimeType:@"image/jpeg" fileName:@"upload.jpg"];

[self.flOperation addCompletionHandler:^(MKNetworkOperation\* operation) {

NSLog(@"%@", [operation responseString]);

**/\***

**This is where you handle a successful 200 response…200 means all worked out…so maybe you make a label that outputs Success!**

**\*/**

}

**//the ^ indicates a block…we’ll get into this later….until then http://developer.apple.com/library/ios/#documentation/cocoa/Conceptual/Blocks/Articles/bxGettingStarted.html#//apple\_ref/doc/uid/TP40007502-CH7-SW1**

errorHandler:^(MKNetworkOperation \*errorOp, NSError\* error) {

NSLog(@"%@", error);

UIAlertView \*alert = [[UIAlertView alloc] initWithTitle:@"Error"

message:[error localizedDescription]

delegate:nil

cancelButtonTitle:@"Dismiss"

otherButtonTitles:nil];

[alert show];

}];

[self.flUploadEngine enqueueOperation:self.flOperation ];

}

**Explaining our upload engine further:**

**To show you how to also pass parameters within the body, there is an dictionary object called postParams.**

**It has one parameter ‘appID’ with the value of ‘testApp’.**

**The network operation, flOperation, is being initialized from our engine’s postDataToServer method. It will pass the postParams dictionary and the POST path.**

**Let’s introduce the compressed image to our network operation. MKNetwork operation has a method called addData that accepts, you guessed it, NSData. Lucky for us, our image is already within an NSData object. Define the success/error blocks and then add the request to the queue. You’re all set!**

**THIS CAN BE USED FOR ALL TYPES OF FILES…video/and audio recordings.**

**Step 4: HANDLE SERVER RESPONSE**

**If you’re following along, character by character, the code will upload a photo and one parameter to mm214.com.**

**Within the response returned as a result of the POST, my server will respond with a URL that will provide us more information about what we just POSTed. Our success block is very basic, outputting the response to the log (viewed within the right section of the bottom pane). If there’s an error with the request, an UIAlertView will appear.**

**You can now add the action created by the button to the implementation file:**

- (IBAction)uploadPhoto:(id)sender {

UIActionSheet \*photoSourcePicker = [[UIActionSheet alloc] initWithTitle:nil

delegate:self cancelButtonTitle:@"Cancel"

destructiveButtonTitle:nil

otherButtonTitles: @"Take Photo",

@"Choose from Library",

nil,

nil];

[photoSourcePicker showInView:self.view];

}

**That’s it….now think of what the switch statement will look like for Video? How about Audio?**