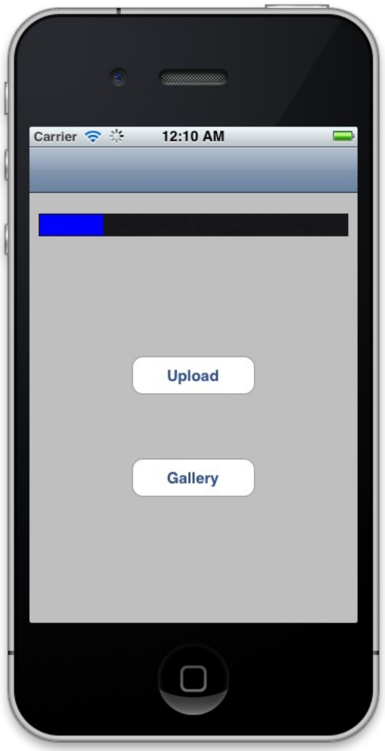
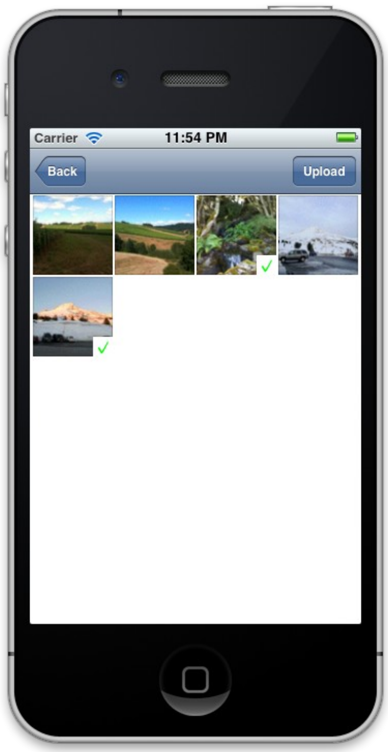
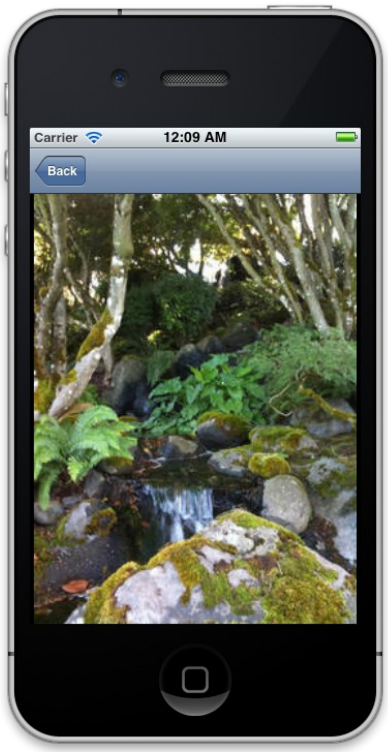
A Chute Image Uploader

This tutorial will walk you through setting up the chute SDK and uploading images from your iOS device to chute. I will also show you how to track the progress of the uploads and add a gallery to your app for viewing the photos added to a chute. This tutorial was written using version 5.0 of the iOS SDK and version 4.2 of Xcode. Some changes may need to be made for other software versions.





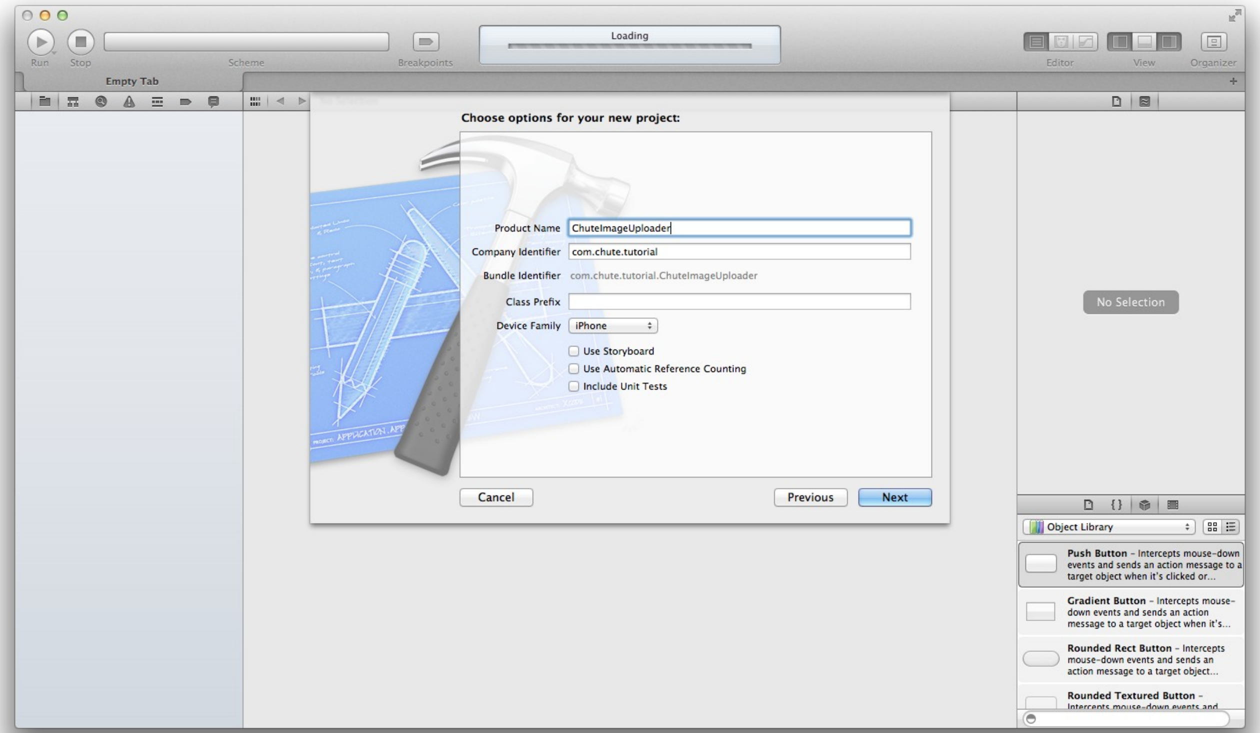


Before We Get Started...

1. Download the Chute SDK from https://github.com/chute/Chute-SDK
2. Download the GCMultiImagePicker and GCCloudGallery components from https://github.com/chute/chute-ios-components
3. Create a Chute developer account and register your app with Chute at http://apps.getchute.com

Creating A New Project

Start by creating a new Xcode project. A single view application will be easiest to modify for this tutorial. You can choose whatever name you like, I'll call it ChuteImageUploader. Be sure that "Use Automatic Reference Counting" is unchecked as the SDK does not currently support ARC.

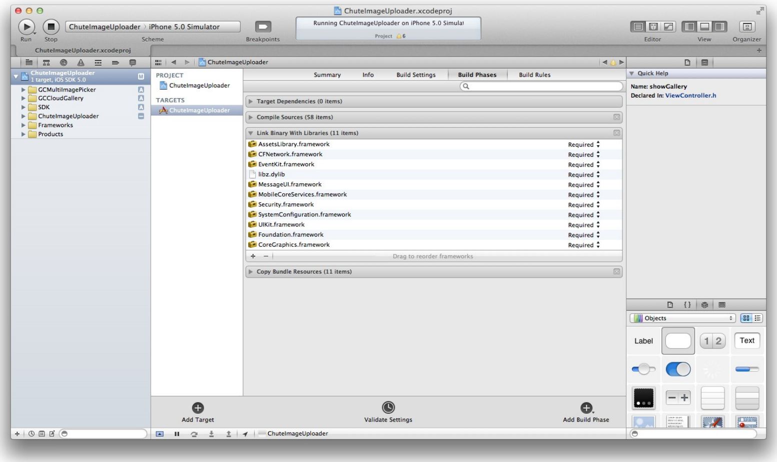


Product Name: ChuteImageUploader
Company Identifier: com.chute.tutorial
Bundle Identifier: com.chute.tutorial.ChuteImageUploader
Class Prefix:
Device Family: iPhone
- [ ] Use Storyboard
- [ ] Use Automatic Reference Counting
- [ ] Include Unit Tests

Push Button – Intercepts mouse-down events and sends an action message to a target object when it's clicked or…
Gradient Button – Intercepts mouse-down events and sends an action message to a target object when it's…
Rounded Rect Button – Intercepts mouse-down events and sends an action message to a target object…
Rounded Textured Button –

Add The SDK And Dependancies

1. Add the SDK and GCImageGrid component to the project
2. Link the required libraries
   - AssetsLibrary
   - CFNetwork
   - EventKit
   - libz.dylib
   - MessageUI
   - MobileCoreServices
   - Security
   - SystemConfiguration



Target Dependencies (0 items)
Compile Sources (58 items)
Link Binary With Libraries (11 items)

AssetsLibrary.framework	Required
CFNetwork.framework	Required
EventKit.framework	Required
libz.dylib	Required
MessageUI.framework	Required
MobileCoreServices.framework	Required
Security.framework	Required
SystemConfiguration.framework	Required
UIKit.framework	Required
Foundation.framework	Required
CoreGraphics.framework	Required

Drag to reorder frameworks
Copy Bundle Resources (11 items)

At this point you may want to try running the project to make sure that everything is added ok. You will get a few warnings, but if there are no errors then everything should be correctly added and linked.

Edit Your App ID And Secret

The next step is to enter your chute client information in the GCConstants.h file. This file can be found in SDK/Classes/Core directory. You will need to fill in your APP ID and APP secret from the settings tab of your admin panel. You will also need to adjust the redirect URL to match the callback url from the admin panel. Then set the redirect relative url to everything after the base in the callback url.

Set Up A Navigation Controller

Next we'll modify the app to use a navigationController. You will need to define a UINavigationController in the appDelegate.h file. Then in the appDelegate.m file you will synthesize the controller and release it in the dealloc method. You will also need to initialize it with your viewController and set it as your window's rootViewController in the application:didFinishLaunchingWithOptions: method. Those changes look like this:

appDelegate.h
```
@property (strong, nonatomic) UINavigationController *navController;
```

appDelegate.m
```
@synthesize navController = _navController;

- (void)dealloc
{
    [_window release];
    [_viewController release];
    [_navController release];
    [super dealloc];
}

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary
*)launchOptions
{
    self.window = [[[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]] autorelease];
    // Override point for customization after application launch.
    self.viewController = [[[ViewController alloc] initWithNibName:@"ViewController" bundle:nil]
autorelease];
    self.navController = [[[UINavigationController alloc]
initWithRootViewController:self.viewController] autorelease];
    self.window.rootViewController = self.navController;
    [self.window makeKeyAndVisible];
    return YES;
}
```

User Login

The first thing we will do is login the user when the app first runs. Import GetChute.h in your viewController.h file to access the Chute SDK. Then simply call the login screen from your viewDidAppear: method. The code for that will look like:

viewController.m
```
[GCLoginViewController presentInController:self];
```

Set Up Your Chute

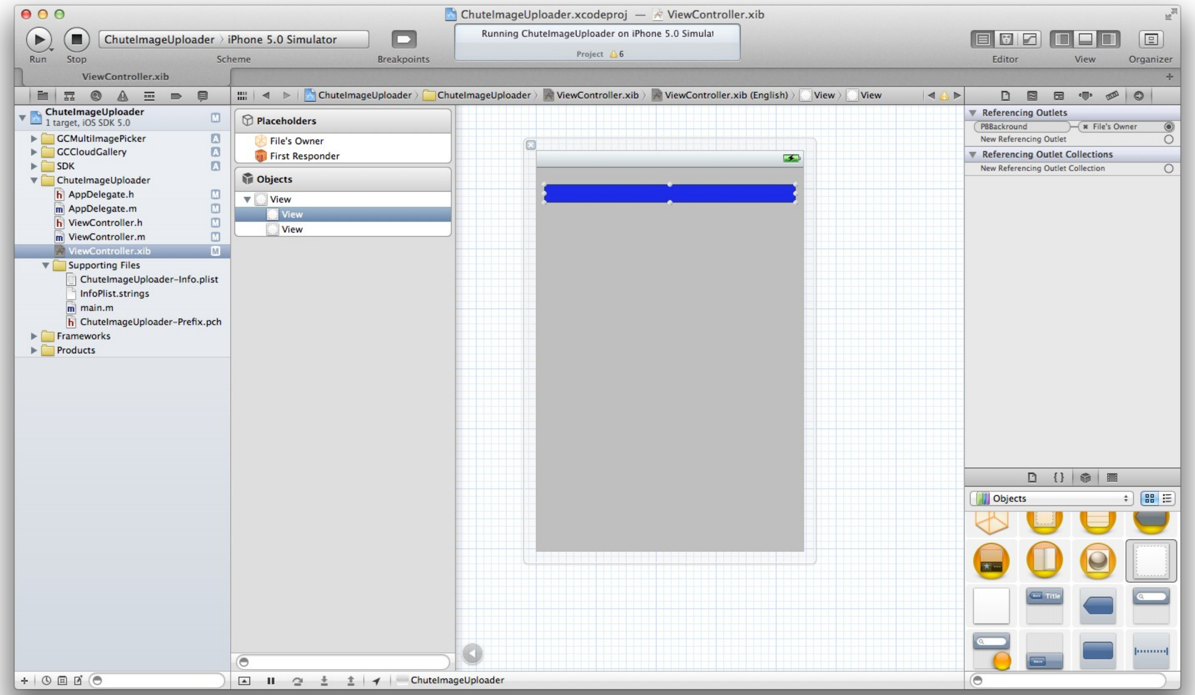
We will need a chute to upload our images to. For this tutorial we will create a chute named Uploads. We will need to set up a GCChute object in our viewController.h file as well as synthesizing and releasing it in the viewController.m file. To make sure that it isn't created multiple times we will save it's id to user defaults and check if it's already created. We also need to make sure the user is already logged in before trying to create the chute. If the chute has been created already we will update the chute object with the latest data. We will also be adding this code to the viewDidAppear: method so it now looks like the following.

viewController.m

```
- (void)viewDidAppear:(BOOL)animated
{
    [super viewDidAppear:animated];

    [GCLoginViewController presentInController:self];

    if([[GCAccount sharedManager] accessToken]){
        if([[NSUserDefaults standardUserDefaults] objectForKey:@"chuteID"]){
            [GCChute findById:[[NSUserDefaults standardUserDefaults] objectForKey:@"chuteID"]
inBackgroundWithCompletion:^(GCResponse *response){
                if([response isSuccessful]){
                    [self setChute:[response object]];
                }
            }];
        }
        else{
            GCChute *_newChute = [GCChute new];

            [_newChute setName:@"Uploads"];
            [_newChute setPermissionView:GCPermissionTypeMembers];
            [_newChute setPermissionAddMembers:GCPermissionTypeMembers];
            [_newChute setPermissionAddPhotos:GCPermissionTypeMembers];
            [_newChute setPermissionAddComments:GCPermissionTypeMembers];
            [_newChute setModeratePhotos:GCPermissionTypePublic];
            [_newChute setModerateMembers:GCPermissionTypePublic];
            [_newChute setModerateComments:GCPermissionTypePublic];

            [_newChute saveInBackgroundWithCompletion:^(BOOL success, NSError *error){
                if(success){
                    [[NSUserDefaults standardUserDefaults] setObject:[_newChute objectID]
forKey:@"chuteID"];
                    [self setChute:_newChute];
                }
            }];
        }
    }
}
```

Create A Progress Bar

Next we will add a progress bar. We will do this in interface builder with a couple of long skinny views. One for the background and one for the foreground of the indicator. First create two IBOutlet UIView objects in your viewController.h file, one called PBForeground and the other PBBackground. Then open the viewController.xib file and add one UIView near the top that is 302px wide by 22 px high. Connect this to the PBBackground object then create another centered on top of it that is 300px by 20px and connect it to PBForeground. Set the background color of these views to whatever you would like.



Add Methods To Control The Progress Bar

We will need three methods for controlling the progress bar. One for when it appears, one for when it disappears and one to update the progress. Use the following code to define these three methods

```
- (void) showProgressIndicator {
    [PBBackground setHidden:NO];
    [PBForeground setHidden:NO];
}

- (void) hideProgressIndicator {
    [PBBackground setHidden:YES];
    [PBForeground setHidden:YES];
}

- (void) progressIndicator:(NSNotification *) notification {
    if ([[GCUploader sharedUploader] progress] > 0 && [[GCUploader sharedUploader] progress] < 1) {
        [self showProgressIndicator];

        [UIView animateWithDuration:0.1 delay:0.0 options:UIViewAnimationOptionAllowUserInteraction
animations:^{
            [PBForeground setFrame:CGRectMake(PBForeground.frame.origin.x,
PBForeground.frame.origin.y, 300*[[GCUploader sharedUploader] progress], 20)];
        } completion:^(BOOL finished) {}];
        return;
    }
    [self hideProgressIndicator];
}
```

Register For Upload Progress Change Notifications

The SDK posts a notification when the upload progress updates. So in the viewDidLoad method we want to initially hide the progressIndicator and add an observer for the progress notification. Therefore we will update the method to look like this

```
- (void)viewDidLoad
```

```
{
    [super viewDidLoad];
    [self hideProgressIndicator];
    [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(progressIndicator:)
name:GCUploaderProgressChanged object:nil];
}
```

Subclass Multi-Image Picker For Upload View

Next we will add the upload view. We will be subclassing the GCMultiImagePicker for this. Add a new class to the project called UploadPicker based on a UIViewController. Be sure that the option for including a xib file is checked. Then modify the UploadPicker.h file to import GCMultiImagePicker.h and inherit from the class. Next we need to add a GCChute object and an upload method. Be sure to synthesize the chute object in your UploadPicker.m file. Once you do this the header file should look like this

UploadPicker.h

```
#import <UIKit/UIKit.h>
#import "GCMultiImagePicker.h"

@interface UploadPicker : GCMultiImagePicker

@property (nonatomic, assign) GCChute *chute;

-(void)uploadSelectedAssets;

@end
```

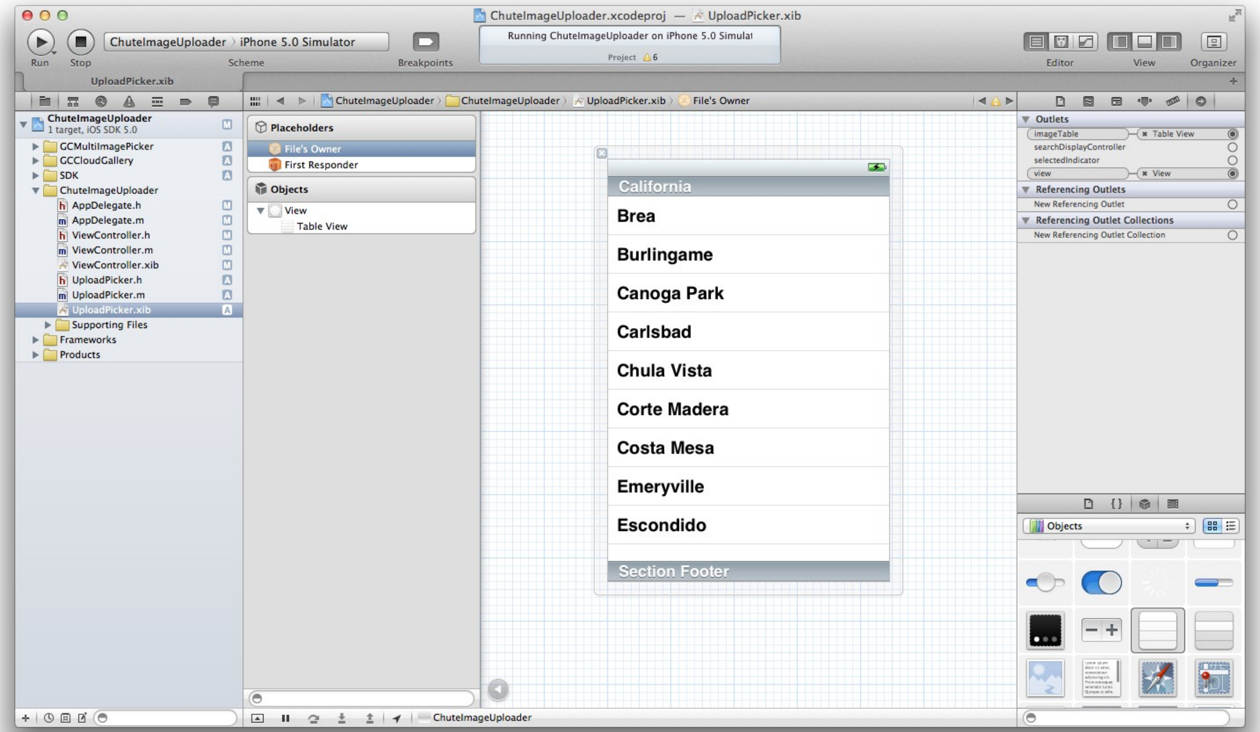
Add Upload Button And Method

Next we need to write the upload method. First we'll do some error checking to make sure there are selected assets. Then we create a parcel with the selected assets and the upload chute. Finally we add it to the upload queue and pop our upload picker from the navigation controller. We also need to set up the viewDidAppear method to show an upload button on the navigation bar that will call this method. These two methods should look like this

```
-(void)uploadSelectedAssets{
    if([[self selectedImages] count] == 0)
        return;
    GCParcel *parcel = [GCParcel objectWithAssets:[self selectedImages] andChutes:[NSArray
arrayWithObject:[self chute]]];
    [[GCUploader sharedUploader] addParcel:parcel];
    [[self navigationController] popViewControllerAnimated:YES];
}

- (void)viewWillAppear:(BOOL)animated
{
    [super viewDidAppear:YES];
    UIBarButtonItem *uploadButton = [[UIBarButtonItem alloc] initWithTitle:@"Upload"
style:UIBarButtonItemStylePlain target:self action:@selector(uploadSelectedAssets)];
    self.navigationItem.rightBarButtonItem = uploadButton;
    [uploadButton release];
}
```

Setup UI For Upload Picker

The only thing left for the upload picker class is the user interface. Open the .xib file and add a UITableView over the entire area. Then connect it to the file owner's imageTable outlet.



Create Button To Show Picker From Main View

Finally we need to put an upload button on the main view and show the upload picker when it's pressed. First import the UploadPicker.h file then add a showUploader method to the viewController.h file with an IBAction return type. In the viewController.m file we will write this method. The method will initialize the UploadPicker, assign the chute to it and push it to the navigation controller. Here is the code for it

```
-(IBAction)showUploader{
    UploadPicker *picker = [[UploadPicker alloc] init];
    [picker setChute:[self chute]];
    [[self navigationController] pushViewController:picker animated:YES];
    [picker release];
}
```

Next add a UIButton to the .xib file and connect it's touchUpInside action to this method. Once this is done you can run the app to test the uploader. Please note that retrieving assets from your device requires location services to be enabled due to location data being associated with ALAssets. If everything is correct then you should be able to upload photos to the chute and watch the upload progress. At this point you won't really be able to see the uploaded images though. Next I'll show you how to add a sliding gallery to view all the assets in the chute.

Create Button To Show Gallery From Main View

Adding the gallery view is simple. We will use the GCCloudGallery component and will not need to subclass or modify it in any way. We will add it basically the same way we added the UploadPicker. Import the GCCloudGallery in the ViewController.h class and add a showGallery method. The method will be slightly different as we need to retrieve the chute assets first then add them to the gallery. The code for this function is

```
-(IBAction)showGallery{
    GCResponse *response = [[self chute] assets];
    if([response isSuccessful]){
        GCCloudGallery *gallery = [[GCCloudGallery alloc] init];
        [gallery setObjects:[response object]];
        [[self navigationController] pushViewController:gallery animated:YES];
        [gallery release];
    }
}
```

After writing the method add a second button to the .xib file and connect the method to it. Finally you can run the app and view the images that have been added to the chute as well as upload more images.

Conclusion

After completing this tutorial you should now be able to use the Chute SDK to create a chute and uploads photos to it. You should also be able to track upload progress and display a gallery view for a chute. I hope you now have a better understanding of the chute SDK and how easy it is to use to manage photos for mobile devices.