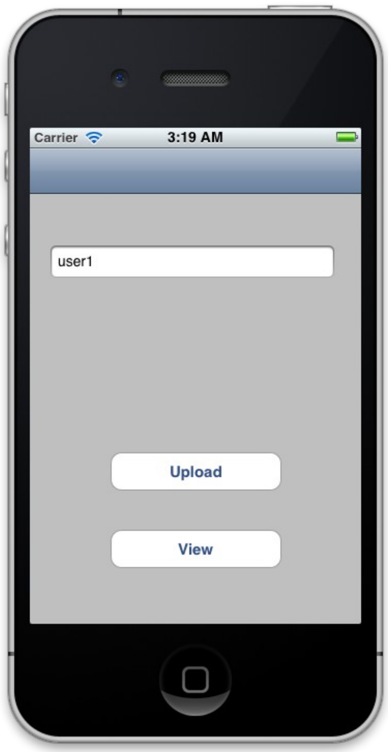
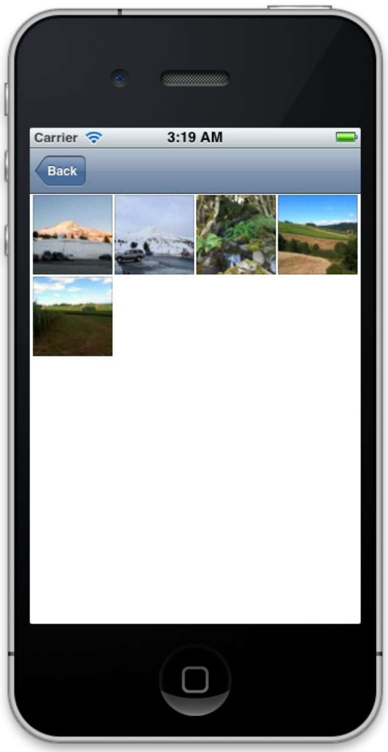
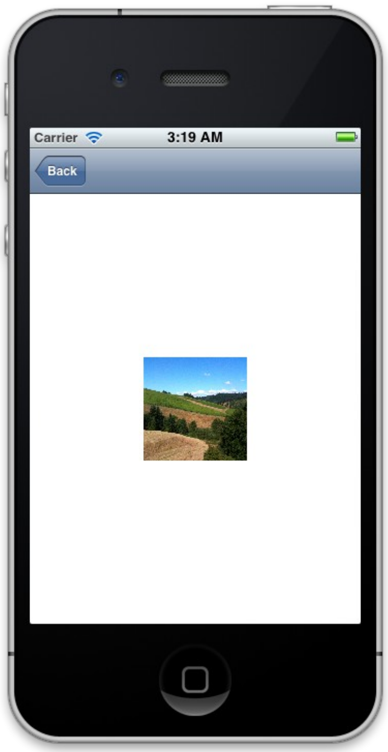
Using Chute to Store User Avatars

This tutorial will show you how to use a chute for storing user avatars. It will show you how to upload an avatar and attach the userID as metadata, as well as how to retrieve an asset using the metadata property. It will also discuss using the admin panel to set up a chute and find the access token and API keys for your app. This tutorial was written using version 5.0 of the iOS SDK and version 4.2 of Xcode. Uses Chute SDK version 1.120115 or newer (the version number can be found in the GCConstants.h file). Some changes may need to be made for other software versions.





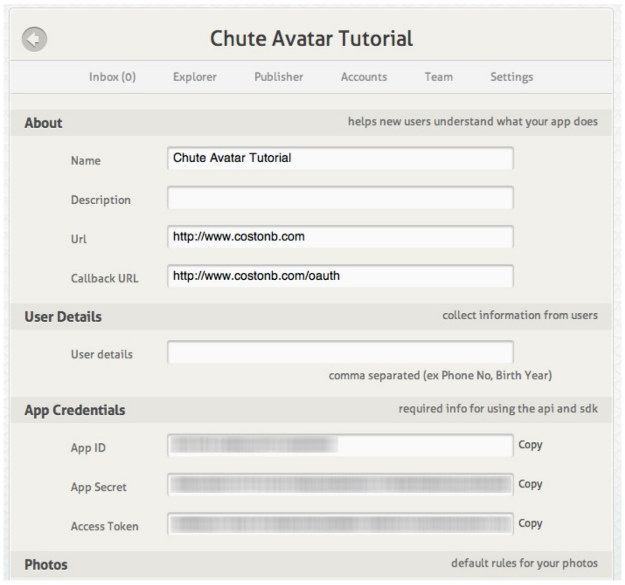


Preparation

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

Create A Chute

After you have created an app you will need to create a chute in it to hold your avatars. This can be done from the chute's online admin panel. Select your app in the list on the chute site and go to the publisher panel. Create a chute with any name that you would like, I called mine Avatars. You can then go to the chutes section of the explorer tab and get the id for the chute you just created. You will need this id later. The settings tab in the admin panel also has information you will need in order to use the SDK. I will go into more detail about that in the next section.



Chute Avatar Tutorial

About — helps new users understand what your app does

Name	Chute Avatar Tutorial
Description	
Url	http://www.costonb.com
Callback URL	http://www.costonb.com/oauth

User Details — collect information from users

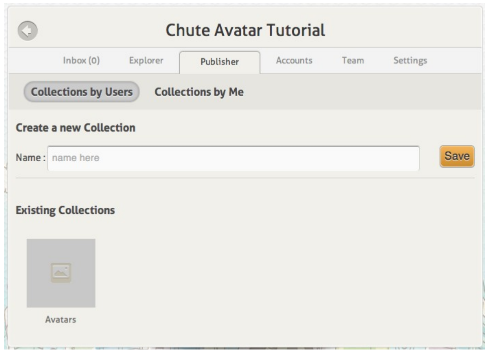
User details	

comma separated (ex Phone No, Birth Year)

App Credentials — required info for using the api and sdk

App ID	[redacted]	Copy
App Secret	[redacted]	Copy
Access Token	[redacted]	Copy

Photos — default rules for your photos



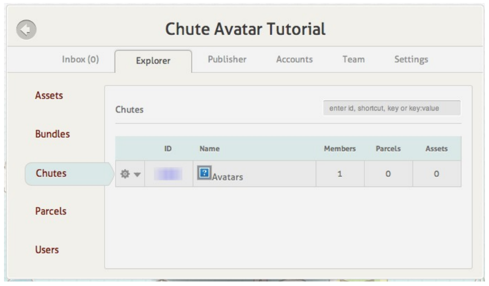
Chute Avatar Tutorial

Create a new Collection

Name : name here

Existing Collections

Avatars



Chute Avatar Tutorial

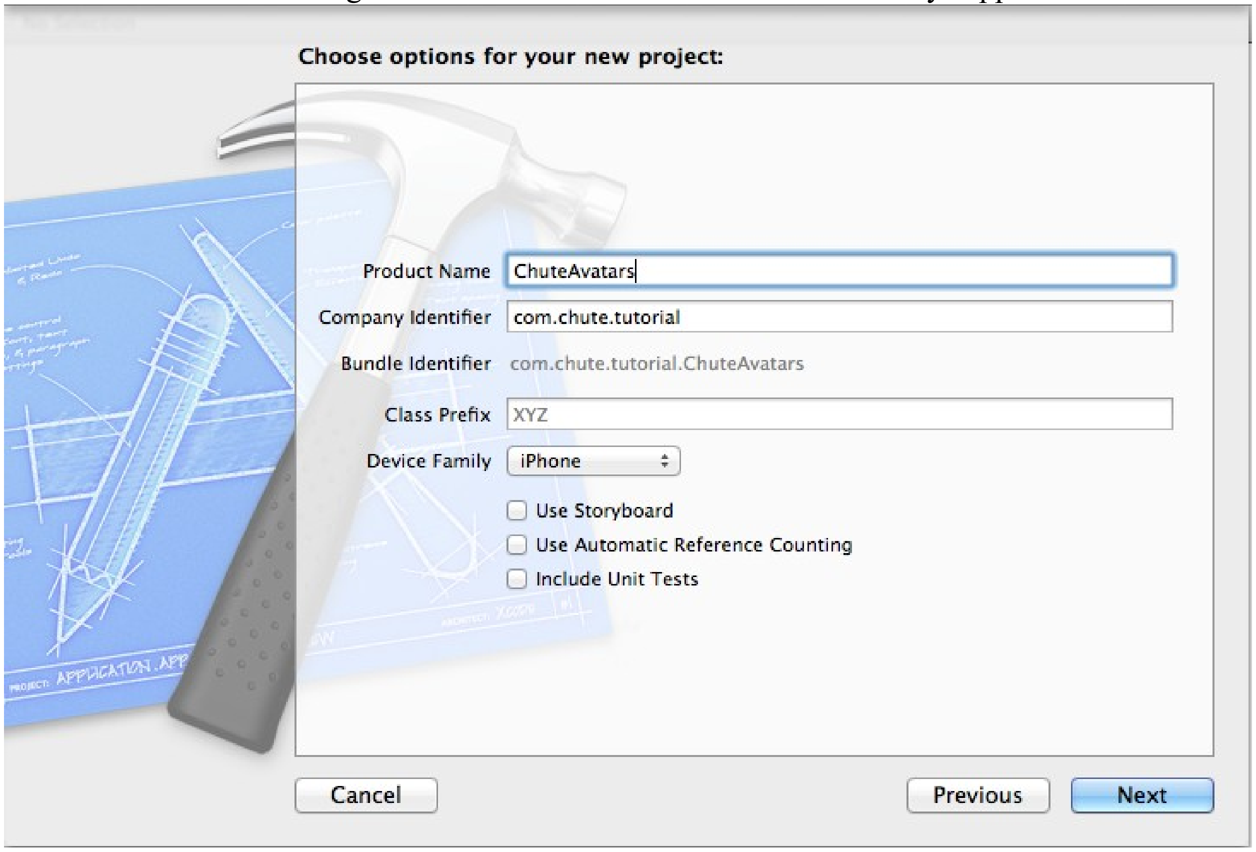
Chutes

enter id, shortcut, key or key:value

	ID	Name	Members	Parcels	Assets
	[redacted]	Avatars	1	0	0

Create 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 ChuteAvatars. Be sure that "Use Automatic Reference Counting" is unchecked as the SDK does not currently support ARC.



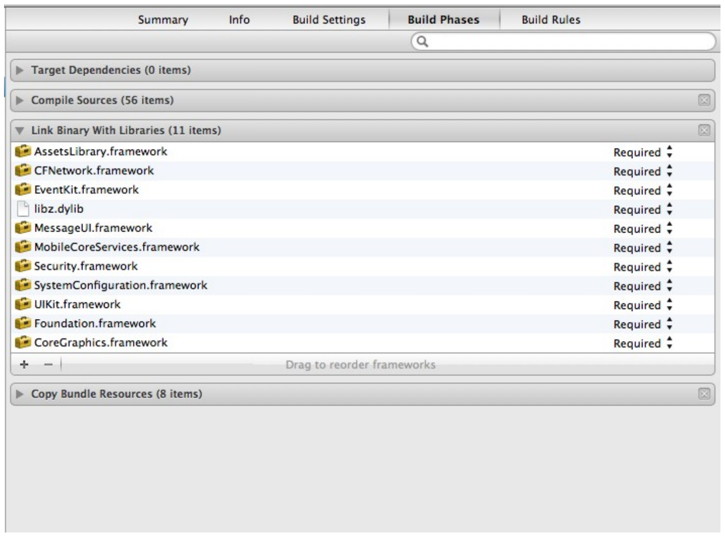
Choose options for your new project:

Product Name	ChuteAvatars
Company Identifier	com.chute.tutorial
Bundle Identifier	com.chute.tutorial.ChuteAvatars
Class Prefix	XYZ
Device Family	iPhone

- [ ] Use Storyboard
- [ ] Use Automatic Reference Counting
- [ ] Include Unit Tests

Add The SDK And Dependancies

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



Summary | Info | Build Settings | Build Phases | Build Rules

Target Dependencies (0 items)

Compile Sources (56 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 (8 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 and set your accessToken. You will need to define a UINavigationController in the appDelegate.h file and also import GetChute.h. Then in the appDelegate.m file you will synthesize the navigation 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. Finally you will set the accessToken object of the GCAccount to the access token from your admin panel. While we are here we will also load the device's assets in the applicationDidBecomeActive: method so that the app has the latest camera roll each time it starts. Those changes look like this (if you are copying and pasting the code, be sure to replace ACCESS_TOKEN with your app's access token):

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
{
    [[GCAccount sharedManager] setAccessToken:@"ACCESS_TOKEN"];
    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;
}

- (void)applicationDidBecomeActive:(UIApplication *)application
{
    [[GCAccount sharedManager] loadAssets];
}
```

Set Up The Main Screen Objects/Methods

The main screen will have a text field to enter a user id, this will simulate retrieving one for your service. Our main screen will also have two buttons, one will bring up a view for selecting a photo, the other will find the user avatar and display it. Both of these will be based on the user id entered in the text field. The main class will also pull the chute that we made in the admin panel. Start with the viewController.h file. Import GetChute.h and add a GCChute object, a GCParcel object and a UITextField object. The text field should be set up as a IBOutlet and the chute and parcel should be retained, synthesized, and released. We will be using the standard UIImagePickerController in this class so you want to inherit from the UIImagePickerControllerDelegate. You can also create two methods for the button presses right now, however we will not write the code for these until later. We will also add in the UITextField delegate method textFieldShouldReturn: and resign the first responder of the text field so that the keyboard will disappear when the enter key is pressed. This should give you the following code:

viewController.h

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

@interface ViewController : GCUIBaseViewController <UITextFieldDelegate,
UIImagePickerControllerDelegate>{
    GCChute *chute;
    GCParcel *parcel;
    IBOutlet UITextField *userID;
}

@property (nonatomic, retain) GCChute *chute;
@property (nonatomic, retain) GCParcel *parcel;

-(IBAction)chooseAvatarClicked:(id)sender;
-(IBAction)viewAvatarClicked:(id)sender;

@end
```

viewController.m

```
@synthesize chute;
@synthesize parcel;

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

-(BOOL)textFieldShouldReturn:(UITextField *)textField{
    [textField resignFirstResponder];
    return YES;
}
```

Retrieve Your Chute

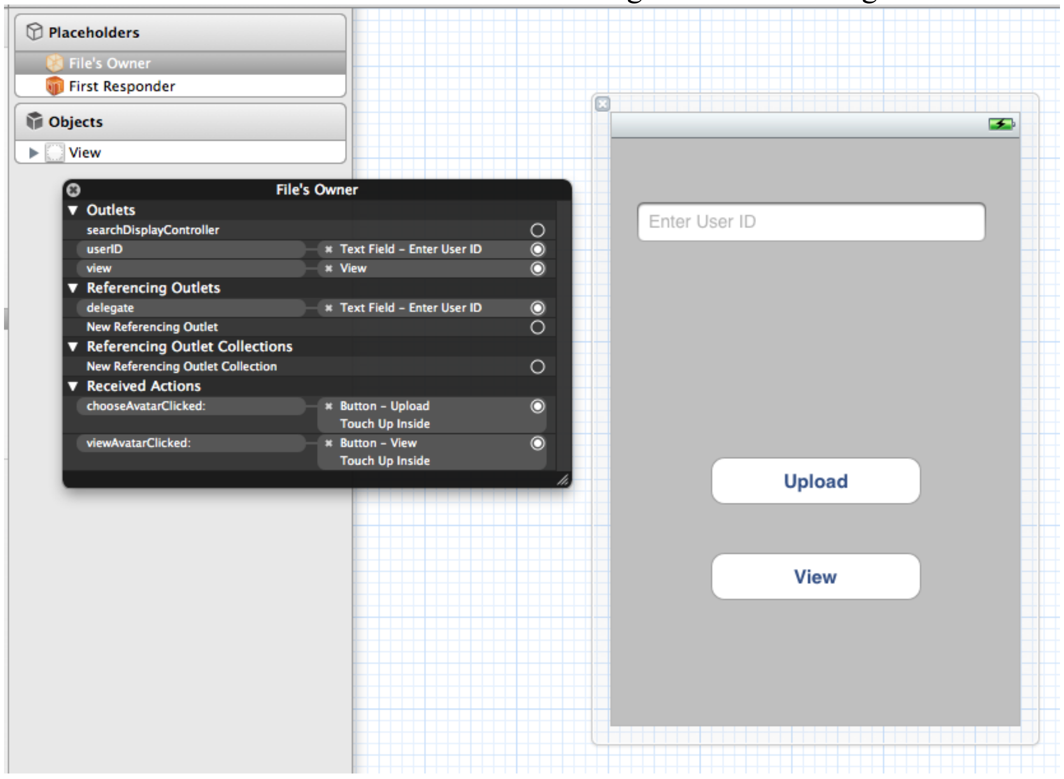
Now we need to retrieve the chute and put in placeholders for the two button press methods in the viewController.m file. For the chute simply call findById and check if the response is successful. Save the response object if it is. This should give you the following code (replace CHUTE_ID with the id for your chute):

viewController.m

```
    -(IBAction)chooseAvatarClicked:(id)sender{

    }
    -(IBAction)viewAvatarClicked:(id)sender{

    }

    - (void)viewDidLoad
    {
        [super viewDidLoad];
        GCResponse *response = [GCChute findById:@"CHUTE_ID"];
        if([response isSuccessful]){
            [self setChute:[response object]];
        }
        else{
            NSLog(@"error retrieving chute");
        }
    }
```

Set Up The Main Screen UI

Finally open the viewController.xib file and place a textField and two buttons. I'm going to use "Enter User ID" for the placeholder text on the text field and "Upload" and "View" for the text on the buttons, but you can use whatever you want. Don't forget to connect the button press methods to the touchUpInside actions for the buttons, connect the text field to it's object and set the file owner as the delegate for the text field. Your view should look something like the following.



Avatar Selection Screen Overview

We will be using the standard UIIMagePickerController for the avatar selection. We will set it up to allow selection from your images on the device and allow editing to be able to select a square section of the photo for the avatar. Once the edited image is selected it will be uploaded to chute and have the user's id attached as metadata.

Show The ImagePicker

We will show our image picker in the chooseAvatarClicked method. We first do some error checking to make sure that there is a valid userID and chute. Then we initialize a UIImagePickerController object and set self as the delegate. We set it to allow editing and set the source type to the photo library. Then we present the controller and release it. The code for this is:

viewController.m
```
-(IBAction)chooseAvatarClicked:(id)sender{
    if(![self chute])
        return;
    if([[[userID text] stringByReplacingOccurrencesOfString:@" " withString:@""] length] == 0)
        return;
    UIImagePickerController *picker = [[UIImagePickerController alloc] init];
    [picker setDelegate:self];
    [picker setAllowsEditing:YES];
    [picker setSourceType:UIImagePickerControllerSourceTypePhotoLibrary];
    [self presentModalViewController:picker animated:YES];
    [picker release];
}
```

Picker Delegate Methods

Next we need to add the UIImagePickerControllerDelegate methods. These are -(void)imagePickerController:(UIImagePickerController *)picker didFinishPickingMediaWithInfo:(NSDictionary *)info and -(void)imagePickerControllerDidCancel:(UIImagePickerController *)picker. The cancel method is simple. All we need to do is dismiss the picker. For the success method we need to dismiss the picker, save the edited image to our library, retrieve the ALAsset, add the image to a parcel and begin the upload process. When we begin the upload process for the parcel we will set self as the delegate and assign a selector called setMetadata that we will create shortly. We also need to save the parcel to our class' parcel object so that we can access it later. The code for all this is the following:

viewController.m

```
-(void)imagePickerController:(UIImagePickerController *)picker didFinishPickingMediaWithInfo:
(NSDictionary *)info{
    [self dismissModalViewControllerAnimated:YES];

    UIImage *originalImage, *editedImage, *imageToSave;

    editedImage = (UIImage *) [info objectForKey:UIImagePickerControllerEditedImage];
    originalImage = (UIImage *) [info objectForKey:UIImagePickerControllerOriginalImage];

    if (editedImage) {
        imageToSave = editedImage;
    } else {
        imageToSave = originalImage;
    }
    if(![[GCAccount sharedManager] assetsLibrary]){
        ALAssetsLibrary *temp = [[ALAssetsLibrary alloc] init];
        [[GCAccount sharedManager] setAssetsLibrary:temp];
        [temp release];
    }
    ALAssetsLibrary *library = [[GCAccount sharedManager] assetsLibrary];
    [self showHUDWithTitle:@"uploading avatar" andOpacity:.75];
    [library writeImageToSavedPhotosAlbum:[imageToSave CGImage] metadata:[info
objectForKey:UIImagePickerControllerMediaMetadata] completionBlock:^(NSURL *assetURL, NSError
*error){
        if(assetURL){
            [library assetForURL:assetURL resultBlock:^(ALAsset* _alasset){

                GCAsset *_asset = [[GCAsset alloc] init];
                [_asset setAlAsset:_alasset];
                GCParcel *_parcel = [GCParcel objectWithAssets:[NSArray arrayWithObject:_asset]
andChutes:[NSArray arrayWithObject:[self chute]]];
                [self setParcel:_parcel];
                [[self parcel] startUploadWithTarget:self andSelector:@selector(setMetadata)];
                [_asset release];
            } failureBlock:^(NSError* error){
            }];
        }
    }];
}
-(void)imagePickerControllerDidCancel:(UIImagePickerController *)picker{
    [self dismissModalViewControllerAnimated:YES];
}
```

Create setMetadata method

Once the upload is complete the parcel will call the setMetadata method. This method will first check if the user has an avatar and if so remove it. Then it will save the user's id to the metadata for the image that was just uploaded. It will get the image by pulling the server assets for the parcel that was just created. Here's what the code for this method looks like:

viewController.m

```
-(void)setMetadata{
    [GCAsset searchMetaDataForKey:@"CAT_USER_ID" andValue:[userID text]
inBackgroundWithCompletion:^(GCResponse *response){
        if([response isSuccessful]){
            NSArray *assetArray = [response object];
            if([assetArray count] > 0){
                for(GCAsset *old in assetArray){
                    [old deleteMetaDataForKey:@"CAT_USER_ID"];
                }
            }
        }
        response = [[self parcel] serverAssets];
        if([response isSuccessful]){
            NSArray *array = [response object];
            if(array.count > 0){
                GCAsset *_asset = [array objectAtIndex:0];
                [_asset setMetaData:[userID text] forKey:@"CAT_USER_ID"
inBackgroundWithCompletion:^(BOOL successful){
                    [self hideHUD];
                    [[self navigationController] setNavigationBarHidden:NO];
                    [[self navigationController] popViewControllerAnimated:YES];
                }];
            }
        }
    }];
}
```

At this point you can try running the app to see if everything is working correctly so far. Please note that your device may ask you to allow location services. This is required when you access ALAssets due to them having location data for the photo associated with them. You won't really be able to see if it uploaded ok yet so in the next section we will see how to pull the image from the server and display it.

Create Avatar Display Screen

The last screen will be for pulling and viewing an avatar that was previously assigned. This screen will just have a UIImageView for the avatar and will be retrieved using the user ID. First we'll add a new file to the app and call it avatarView. This class will need a NSString object for the user ID and an IBOutlet UIImageView object. We also need to import GetChute.h. This gives us a header file that looks like:

avatarView.h

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

@interface avatarView : UIViewController

@property (nonatomic, assign) NSString *userID;
@property (nonatomic, readonly) IBOutlet UIImageView *avatar;

@end
```

Find User Avatar

Next we will synthesize the objects and modify the viewDidAppear method to pull the avatar and display it. First we need to find the Asset associated with the user id so we will search for the metadata key/value pair. Once we find it we can use the asset url to display the avatar. The Chute SDK includes a file that extends the UIImageView class to add web caching. We will use this to simplify displaying the avatar. The code for all this is:

avatarView.m

```
@synthesize userID, avatar;

-(void)viewDidAppear:(BOOL)animated{
    [super viewDidAppear:animated];
    [GCAsset searchMetaDataForKey:@"CAT_USER_ID" andValue:[self userID]
inBackgroundWithCompletion:^(GCResponse *response){
        if([response isSuccessful]){
            NSArray *assets = [response object];
            if([assets count] > 0){
                GCAsset *asset = [assets objectAtIndex:0];
                [[self avatar] setImageWithURL:[NSURL URLWithString:[asset
urlStringForImageWithWidth:100 andHeight:100]]];
            }
        }
    }];
}
```

Set Up Avatar Display UI

To set up the nib file we will simple put a UIImageView in the middle of the view and hook it up to the avatar object of the file owner. I set the image view to 100px by 100px and removed all autoresizing so that it would stay the same size and centered in the middle of the view. You can set this to whatever you'd like.

Connect Avatar Display to MainScreen

The final step is to fill in the viewAvatarClicked method of the viewController class. This method will do some error checking to make sure a user id is entered. Then it will initialize a avatarView object and set the user id object. Finally it will push the avatar view to the navigationController and release the avatarView object. The code for this is (Don't forget to import your avatarView.h file):

viewController.m

```
-(IBAction)viewAvatarClicked:(id)sender{
    if([[[userID text] stringByReplacingOccurrencesOfString:@" " withString:@""] length] == 0)
        return;
    avatarView *av = [[avatarView alloc] init];
    [av setUserID:[userID text]];
    [[self navigationController] pushViewController:av animated:YES];
    [av release];
}
```

Conclusion

You should now have a fully functioning app that will save user avatars, save a user id in the metadata, and retrieve the avatar using the metadata. In an actual app you could make a few improvements. If you have persistent data for your users either in the app or on your servers you could save the objectID for the asset that represents their avatar. This would cut down on the server calls and make your app more efficient. Then when you need to retrieve the avatar asset you can use the findByID method of the GCAsset class. Another possibility would be to save the image url directly which would cut out one more server call which helps speed up your app. The Chute SDK is flexible enough that it should be able to accommodate whatever your needs are. This tutorial is merely a starting point and reference for maintaining your user's avatars. Thank you for reading it and I look forward to seeing what you are able to achieve with our platform.