Using Chute to Store User Avatars

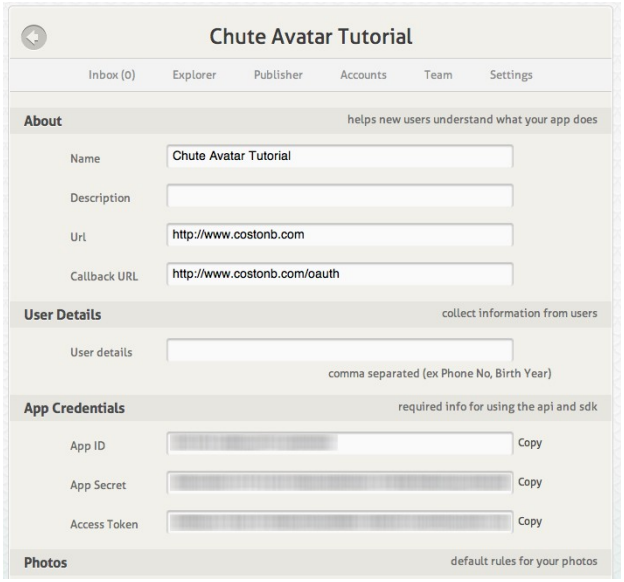
This tutorial will show you how to use a chute for storing user avatars. It will show you how to upload an avatar and attach the userID as metadata, as well as how to retrieve an asset using the metadata property. It will also discuss using the admin panel to set up a chute and find the access token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

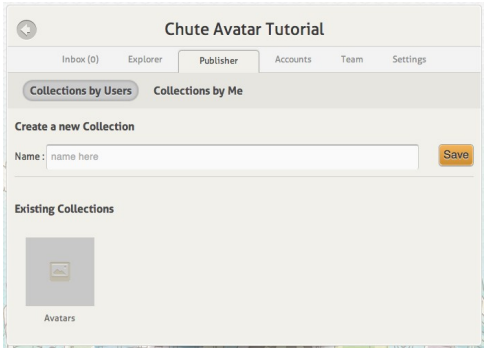
Preparation

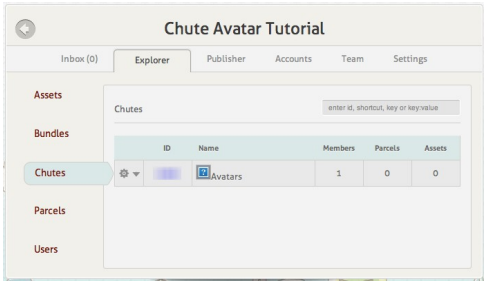
1. Download the Chute SDK from https://github.com/chute/Chute-SDK
2. Download the GCImageGrid component from https://github.com/chute/chute-ios-components
3. Create a Chute developer account and register your app with Chute

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.

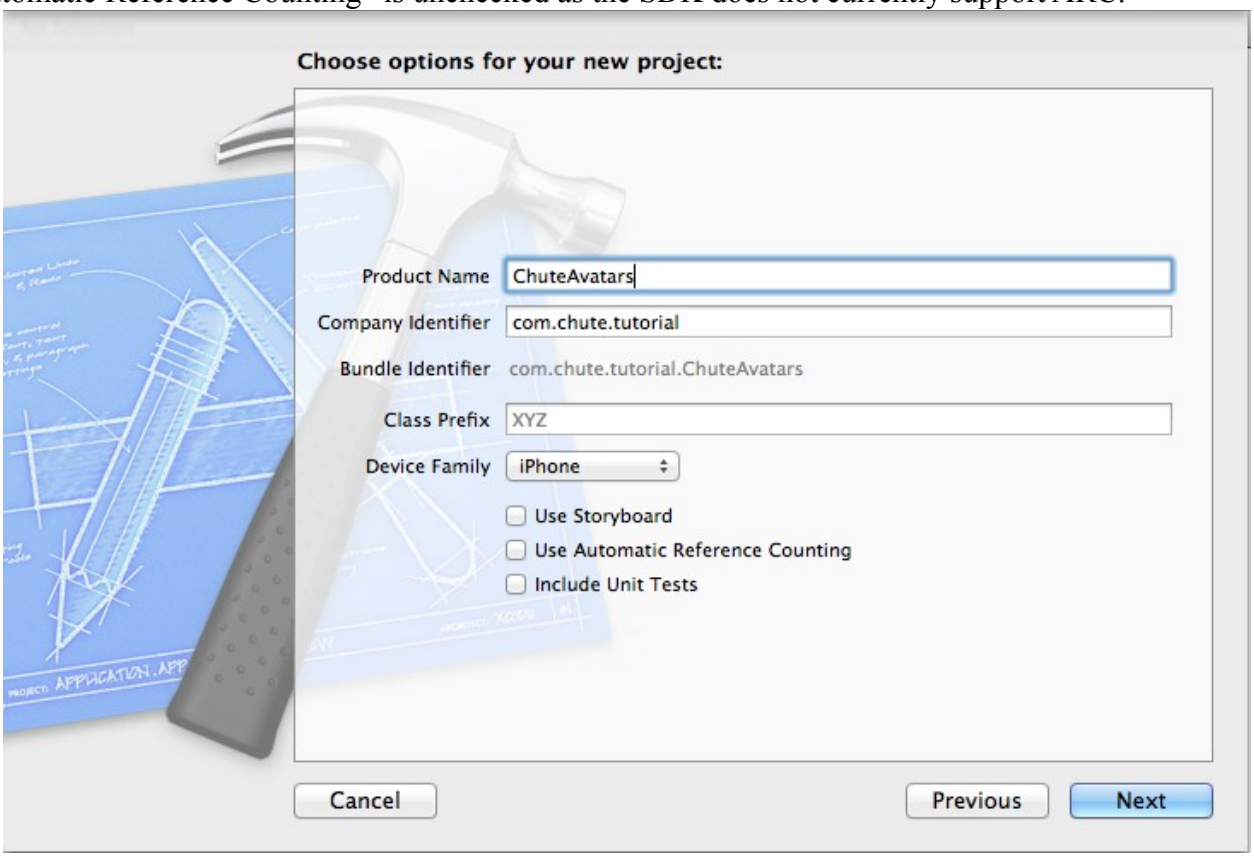






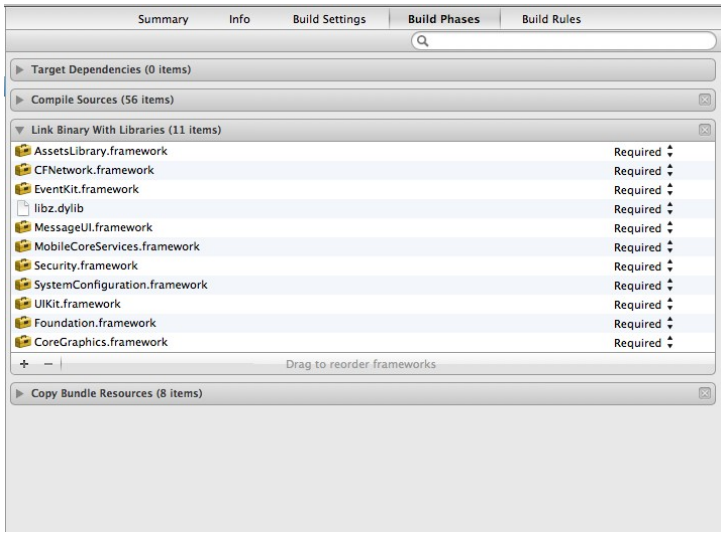
Create A New Project

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.



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



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

Will 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 and a UITextField object. The text field should be set up as a IBOutlet and the chute should be retained, synthesized, and released. 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 : UIViewController{
    GCChute *_chute;
    IBOutlet UITextField *userID;
}

@property (nonatomic, retain) GCChute *chute;

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

@end
```

viewController.m

```
@synthesize chute = _chute;

-(void)dealloc{
    [_chute 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

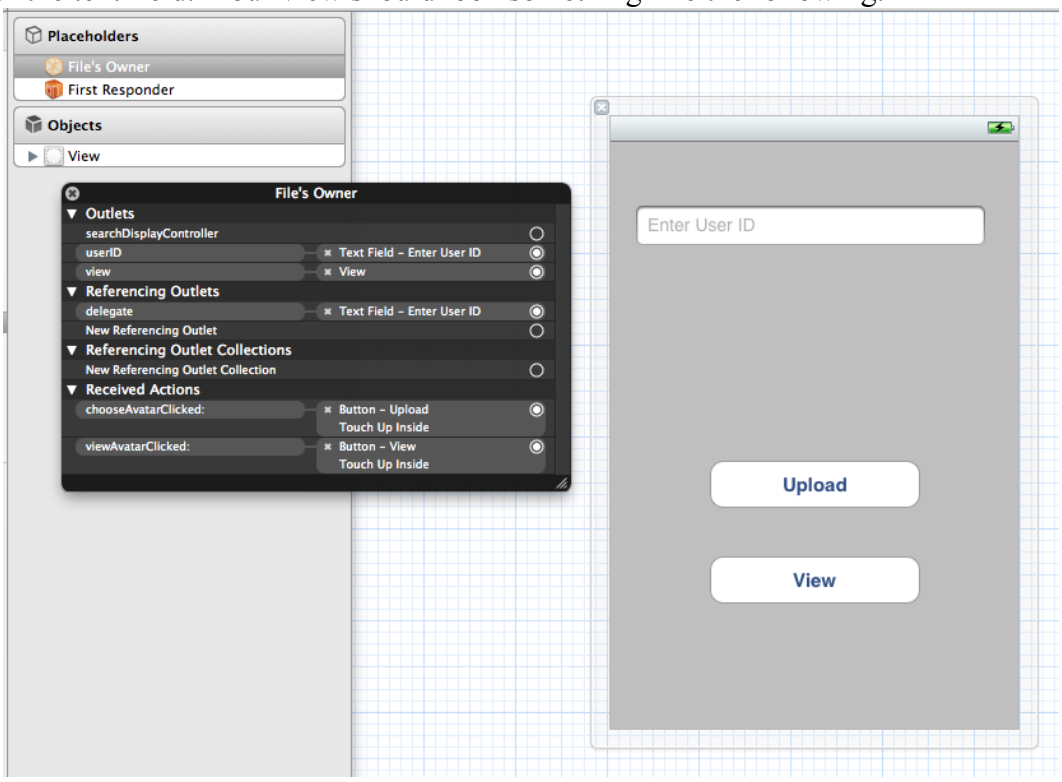
the viewController.m file. For the chute simply call findById and check if the response is successful. Save the response object if it is. This should give you the following code (replace CHUTE_ID with the id for your chute):

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

The avatar selection view will be a subclass of the GCImageGrid class. This class normally has

a grid of GCAssets and tapping on one displays it full screen. We will overwrite the objecctTappedAtIndex: method in our subclass to instead upload the image to a chute and add the user id as a metadata object then exit the screen. Our subclass will need objects for the chute to upload to, the user id it needs to save and the asset that will be uploaded. We will also need to supply it with the assets from the device's camera roll.

Subclass GCImageGrid

First create a new class based on a UIViewController. I called mine imagePicker. In the imagePicker.h file import GCImageGrid.h and modify the class to inherit from GCImageGrid rather than the UIViewController class. Then add a NSString object for the user id, a GCChute object for the upload chute and a GCAsset object. The asset will need to be retained, however you can just set the other objects to assign. This gives us a header file that looks like this:

imagePicker.h

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

@interface imagePicker : GCImageGrid

@property (nonatomic, assign) NSString *userID;
@property (nonatomic, assign) GCChute *chute;
@property (nonatomic, retain) GCAsset *asset;

@end
```

Write Upload And MetaData Functions

In the imagePicker.m file we need to synthesize our objects and set up a dealloc method to release the parcel. Then we can overwrite the objectTappedAtIndex method. This method will create a parcel from the selected asset and the chute. Since we need custom behavior after uploading we will not be using the GCUploader class to manage the uploads. Instead we will start the upload directly in the parcel and assign a method to be executed when it's done. This method will check if there is already an asset with that userID. If it finds one it will remove it. It then adds the metadata and pops the view from the navigation controller. We will display an activity indicator while this is going on that will inform the user that it is updating their avatar. The GCImageGrid class is based on the GCUIBaseViewController which has a method for easily presenting an indicator to the user. The code for all this is:

imagePicker.m

```
@synthesize userID, chute, asset;

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

-(void)setMetadata{
    [GCAsset searchMetaDataForKey:@"CAT_USER_ID" andValue:[self userID]
inBackgroundWithCompletion:^(GCResponse *response){
        if([response isSuccessful]){
            NSArray *assetArray = [response object];
            if([assetArray count] > 0){
                for(GCAsset *old in assetArray){
                    [old deleteMetaDataForKey:@"CAT_USER_ID"];
                }
            }
        }
        NSDictionary *objectData = [[[[self asset] verify] object] objectAtIndex:0];
        [self setAsset:[GCAsset objectWithDictionary:objectData]];
        [[self asset] setMetaData:[self userID] forKey:@"CAT_USER_ID"
inBackgroundWithCompletion:^(BOOL successful){
            [self hideHUD];
            [[self navigationController] setNavigationBarHidden:NO];
```

```
            [[self navigationController] popViewControllerAnimated:YES];
        }];
    }];
}

-(void)objectTappedAtIndex:(NSInteger)index{
    [[self navigationController] setNavigationBarHidden:YES];
    [self hideHUD];
    [self showHUDWithTitle:@"Uploading Avatar" andOpacity:.7];
    [self setAsset:[objects objectAtIndex:index]];
    GCParcel *parcel = [GCParcel objectWithAssets:[NSArray arrayWithObject:[self asset]] andChutes:
[NSArray arrayWithObject:[self chute]]];
    [parcel startUploadWithTarget:self andSelector:@selector(setMetadata)];
}
```

Set Up ImagePicker UI

Next we will set up the nib file. This is extremely easy for this class. Simply fill the view with a UITableView and hook it up to the objectTable in the file's owner.

Connect ImagePicker To MainScreen

Now we only need to fill in the chooseAvatarClicked method and import our imagePicker.h file in viewController.m. For the method we will first do some error checking to make sure we have a chute and user id. Then we initialize an imagePicker object and set it's chute and user ID. Next we set it's objects to our device assets. Finally we push it to the navigation controller and release our object. The code for this looks like:

viewController.m

```
-(IBAction)chooseAvatarClicked:(id)sender{
    if(![self chute])
        return;
    if([[[userID text] stringByReplacingOccurrencesOfString:@" " withString:@""] length] == 0)
        return;
    imagePicker *picker = [[imagePicker alloc] init];
    [picker setChute:[self chute]];
    [picker setUserID:[userID text]];
    [picker setObjects:[[GCAccount sharedManager] assetsArray]];
    [[self navigationController] pushViewController:picker animated:YES];
    [picker release];
}
```

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.