KANDY IOS SDK 入门

中文版

上海甜辛科技有限公司

目录

1. 创建开发者账号 2

1.1. 账号、项目、用户关系 2

1.2. 申请账号 2

1.3. 登陆开发者网站 3

1.4. 开发者后台创建用户 5

2. IOS SDK 快速入门 6

2.1. IOS SDK下载 6

2.2. 运行SampleSDKApp工程 7

2.3. SampleSDKApp拨打电话 9

3. IOS SDK使用 10

3.1. 如何使用本文档 10

3.2. 开始之前 10

3.3. 用户注册 14

3.4. 用户登录 16

3.5. VOIP 推送 19

3.5.1. 前提条件 19

3.5.2. VOIP推送 19

3.5.3. 开启推送功能 19

3.5.4. 处理推送 19

3.6. 定位 20

3.6.1. 国家信息 20

3.7. 呼叫 21

3.7.1. 呼叫设置 21

3.7.2. 发起Voip呼叫 21

3.7.3. 发起Voip到PSTN呼叫 21

3.7.4. 呼叫状态监听 21

3.7.5. 收到呼叫操作 21

3.7.6. 呼叫统计 21

3.8. 多人会议 22

3.8.1. 加入房间 22

3.8.2. 离开房间 22

3.8.3. 销毁房间 22

3.8.4. 获取会议状态 22

3.8.5. 更新参会者姓名 22

3.8.6. 会议控制 22

3.9. 聊天 23

3.9.1. 聊天状态监听 23

3.9.2. 发送文字消息 23

3.9.3. 发送组消息 23

3.9.4. 发送富媒体消息 23

3.9.5. 发送制定类型事件 23

3.9.6. 负载自定义消息体 23

3.9.7. 发送短信消息 23

3.9.8. Pull事件 23

3.9.9. 确认事件 23

3.9.10. 下载媒体消息 23

3.9.11. 下载媒体封面 23

3.9.12. 取消媒体操作 23

3.9.13. 设置相关 23

3.10. 群组消息 24

3.10.1. 监听群组状态 24

3.10.2. 创建群组 24

3.10.3. 添加／删除群组成员 24

3.10.4. 群组状态更新 24

3.10.5. 销毁群组 24

3.10.6. 我的群组 24

3.10.7. 群组详情 24

3.10.8. 群组设置 24

3.11. 服务器事件 25

3.11.1. 监听服务器事件 25

3.11.2. Pull Pending 事件 25

3.11.3. 确认Pending 事件 25

3.12. 在线状态 26

3.12.1. 最近在线状态 26

3.12.2. 更新在线状态 26

3.12.3. 开始监听在线状态 26

3.12.4. 结束监听在线状态 26

3.13. 通讯录管理 27

3.13.1. 监听通讯录状态 27

3.13.2. 获取设备的通讯录 27

3.13.3. 查询设备通讯录 27

3.13.4. 检索域通讯录 27

3.13.5. 查询域通讯录 27

3.13.6. VCard名片 27

3.14. 云存储 28

3.14.1. 上传文件 28

3.14.2. 下载文件 28

3.14.3. 下载封面 28

3.14.4. 取消操作 28

3.15. 错误处理 29

|  |  |  |  |
| --- | --- | --- | --- |
| 时间 | 版本 | 更新内容 | 作者 |
| 2017/1/23 | 1.0 | 初始版本 | Quantong.ai |
| 2017/1/25 | 1.0 | 修正版本 | Quantong.ai |
| 2017/2/5 | 1.0 | 初始版本完成 | Quantong.ai |
|  |  |  |  |

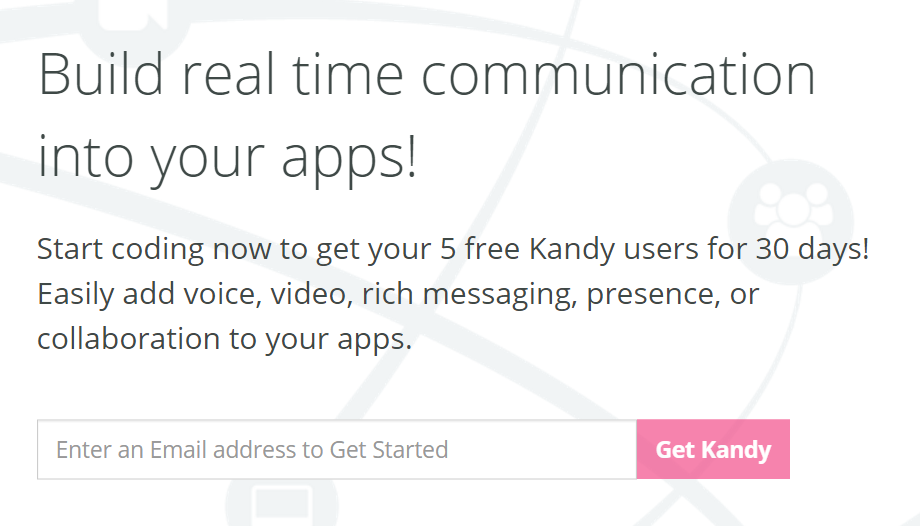
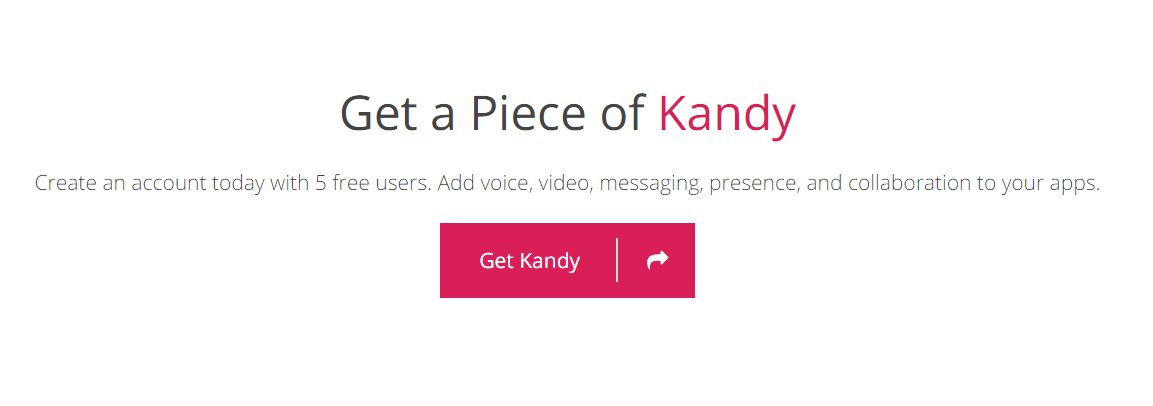
# 创建开发者账号

## 账号、项目、用户关系

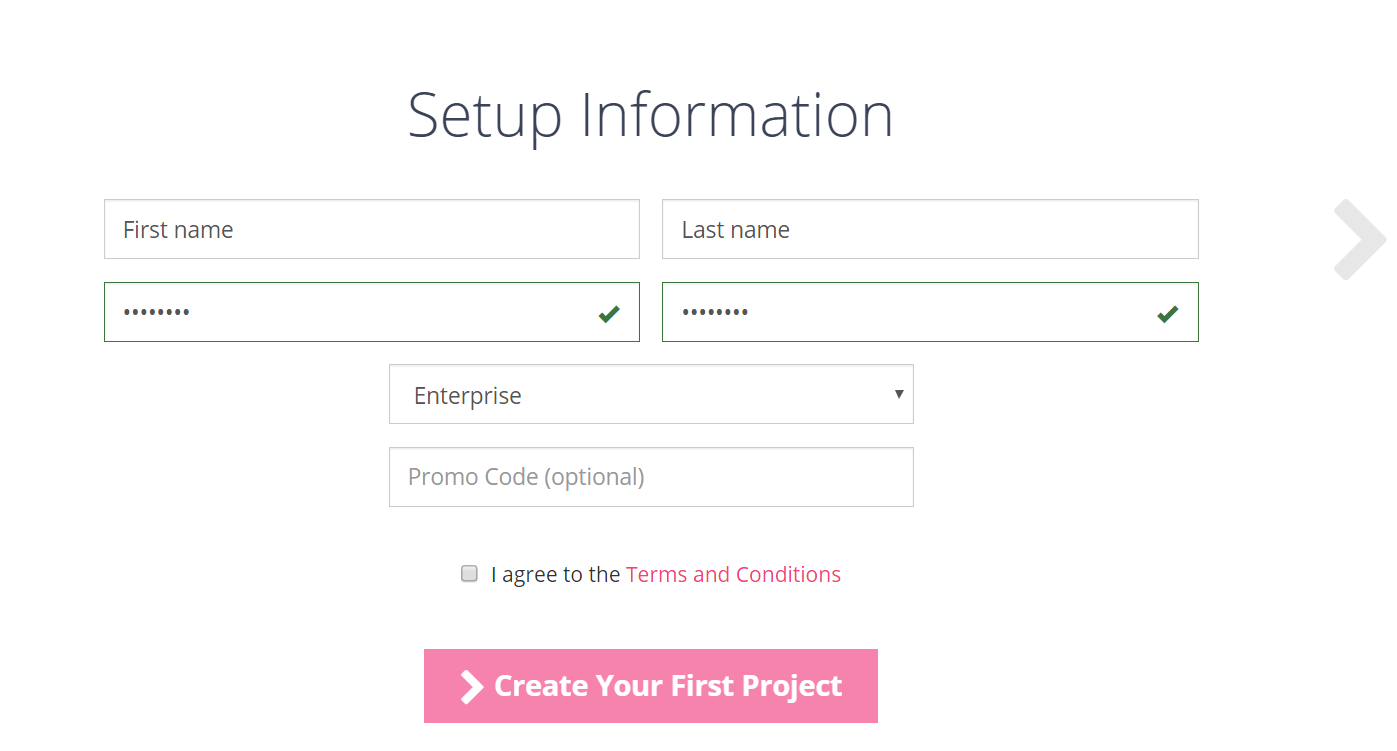
账号、项目、用户这三个概念是有层级关系的：账号下可以创建多个项目，每个项目下可以创建用户，用户作用在SDK上实现相关功能。

## 申请账号

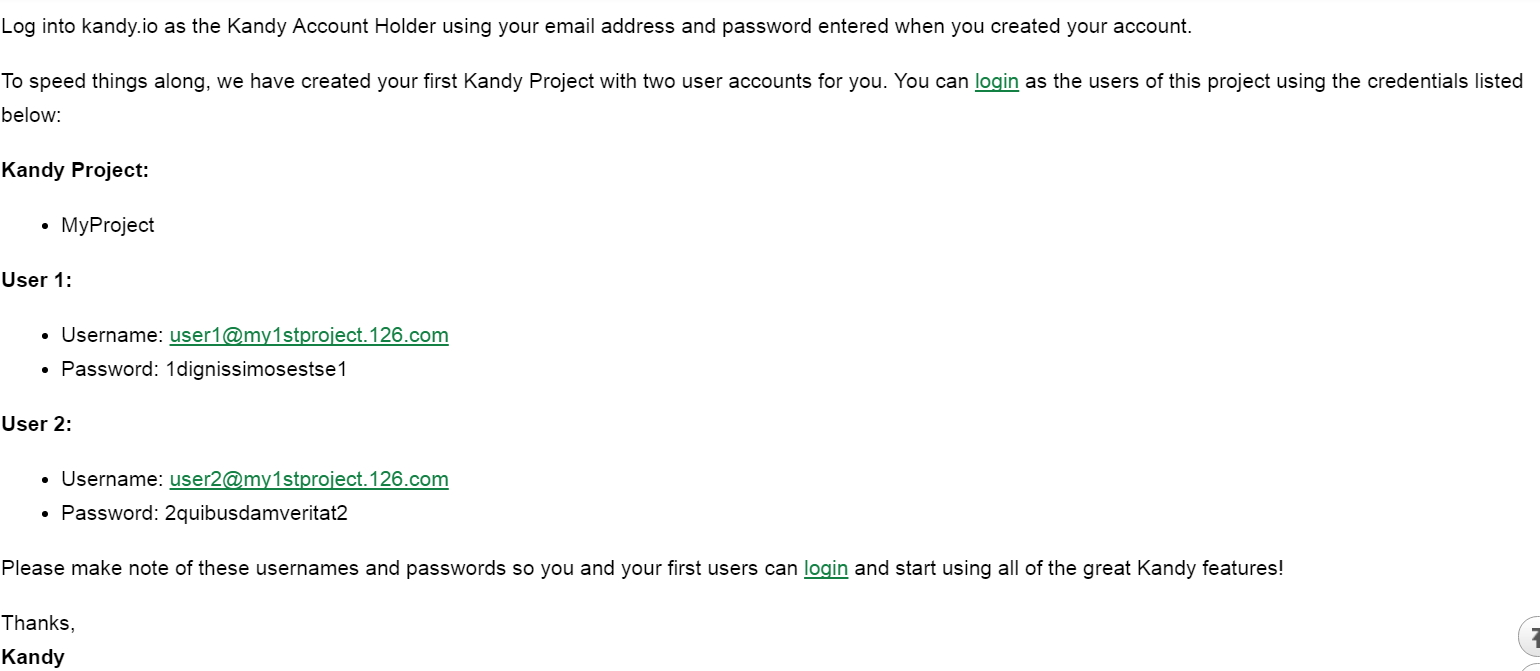
在使用Kandy前，需要在官网（https://www.kandycn.com/）上填入您自己的邮箱注册账号。



在邮箱中，将会收到系统给您发送的注册确认邮件，点击邮件上的确认连接，将要求您输入您的密码和简单个人信息。

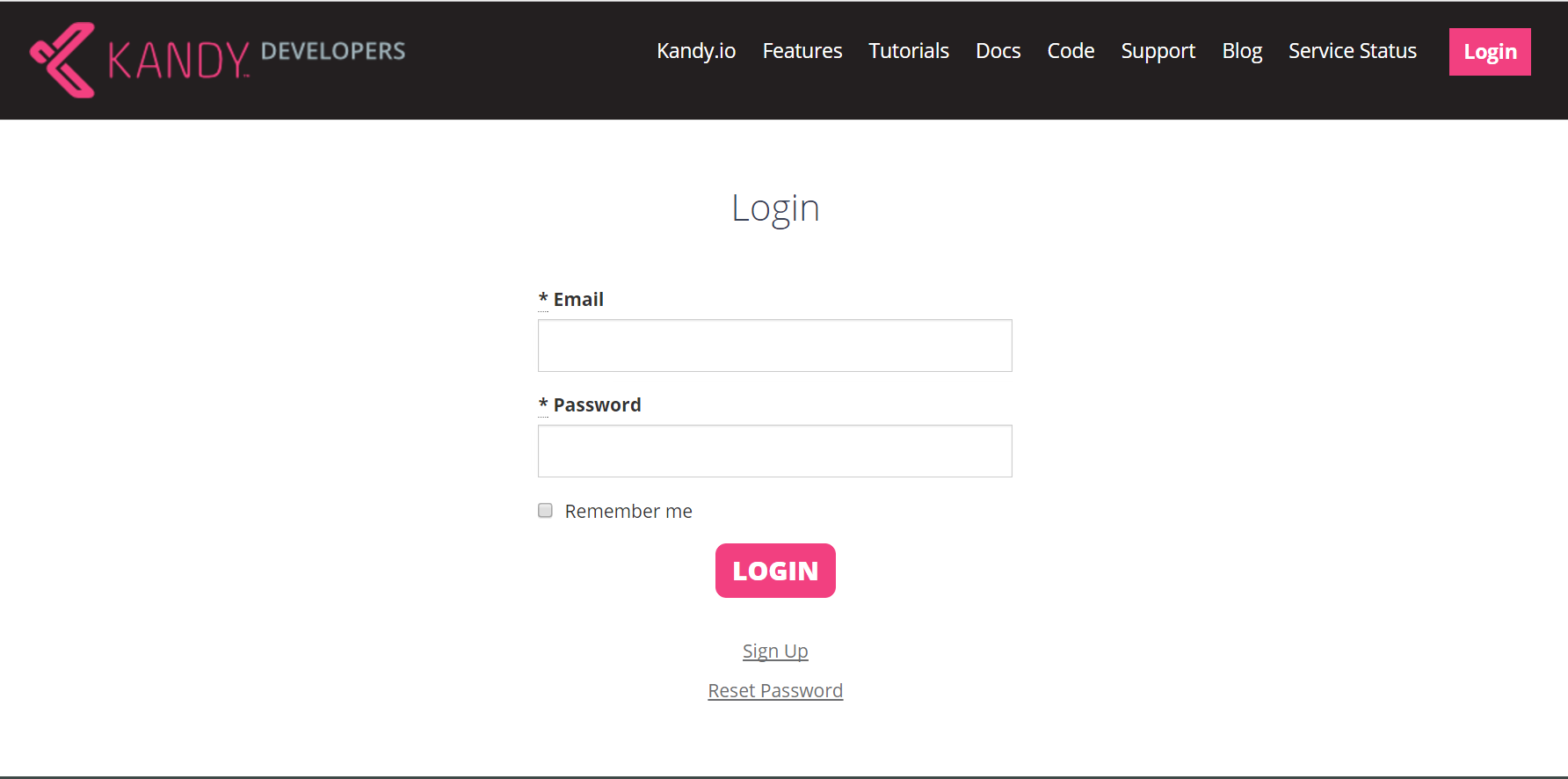


如果确认注册成功，系统将默认为您创建一个项目，并且在这个项目下有两个用户，相关信息会以邮件方式通知您。

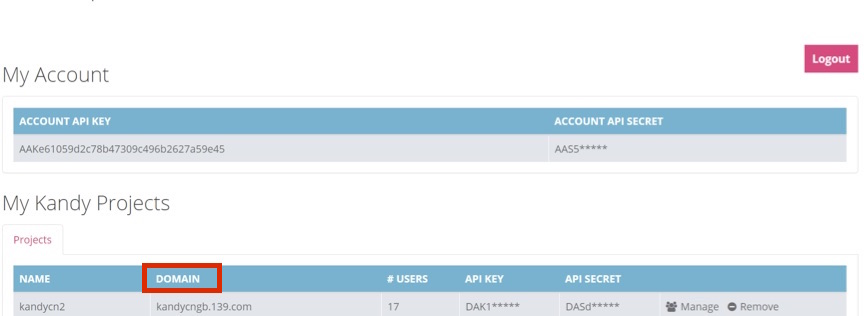


## 登陆开发者网站

当您的账号申请成功后，您就能够用账号名和密码来登陆开发者网站- https://developer.kandycn.com/login，管理您的项目和其下面的用户。



登陆您账号的管理页面后，您将看到您的账号的key和Secret，以及之前您所创建的项目的Key和Secret。账号和项目的Key和Secret是十分重要的信息，请妥善保管好这两个信息。



* Key和Secret

账号级别的API操作，例如：项目的创建、删除等，需要账号的Key，Secret。

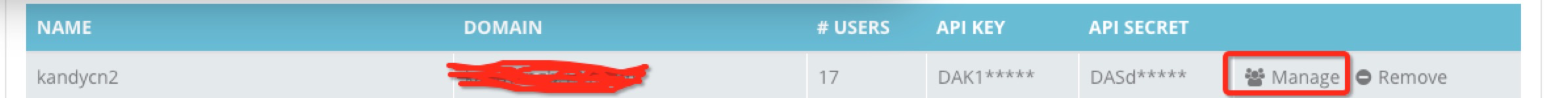
项目级别的功能，例如：SDK使用、用户的管理（创建、删除）等，需要项目的Key，Secret。

* 特别说明［域－DOMAIN］

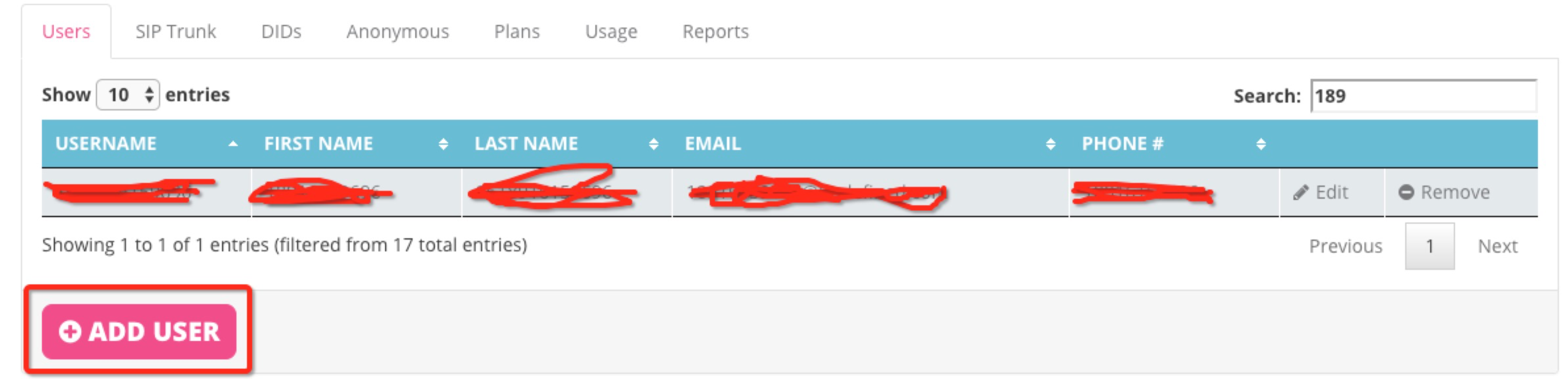
Kandy的账户系统是通过域来进行管理，域可用来标示账号来源不同的APP Key，也要通过创建子域而达到分地区的管理用户的目标。

## 开发者后台创建用户

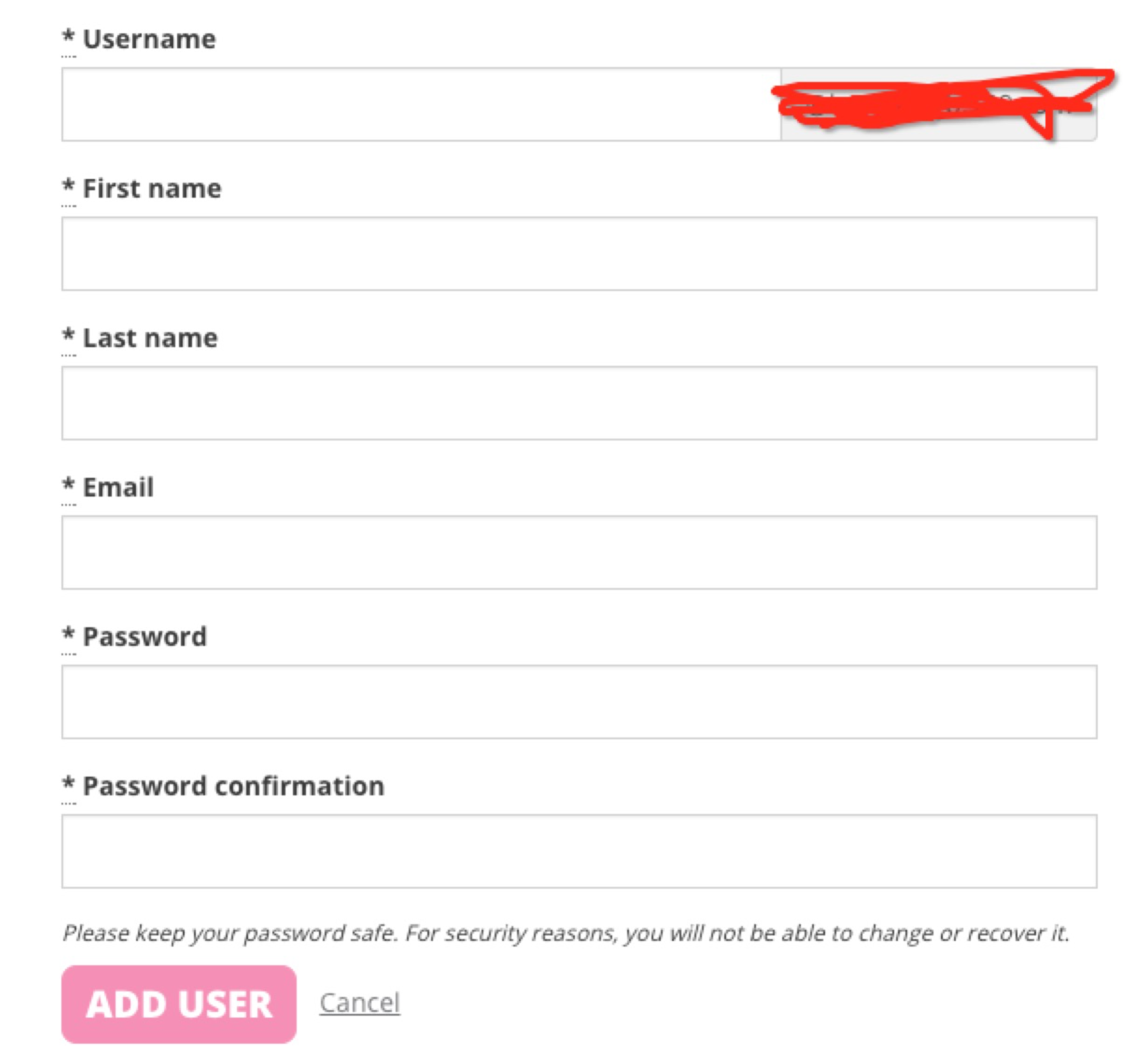
点击您的账号对应的app key 的Manage



点击ADD USER 也可以点击Edit 对以创建的用户进行编辑



输入用户名和密码创建用户

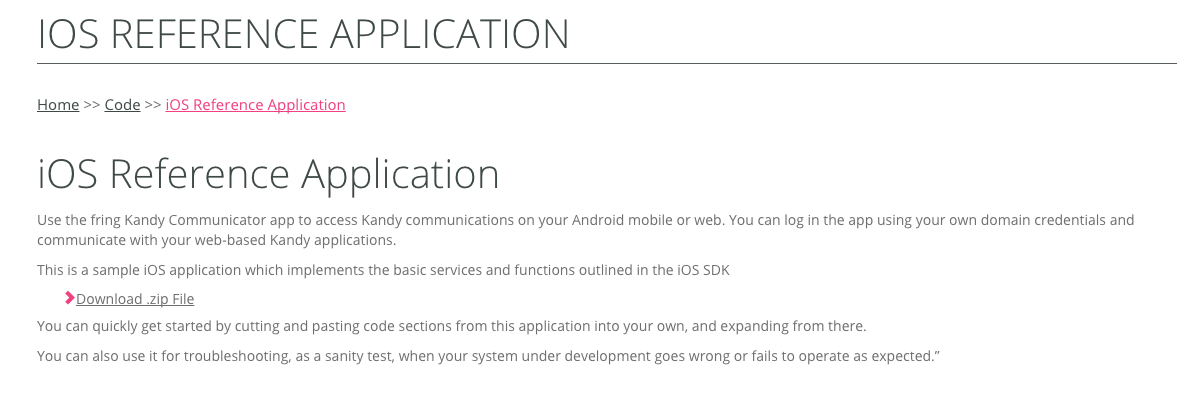


# IOS SDK 快速入门

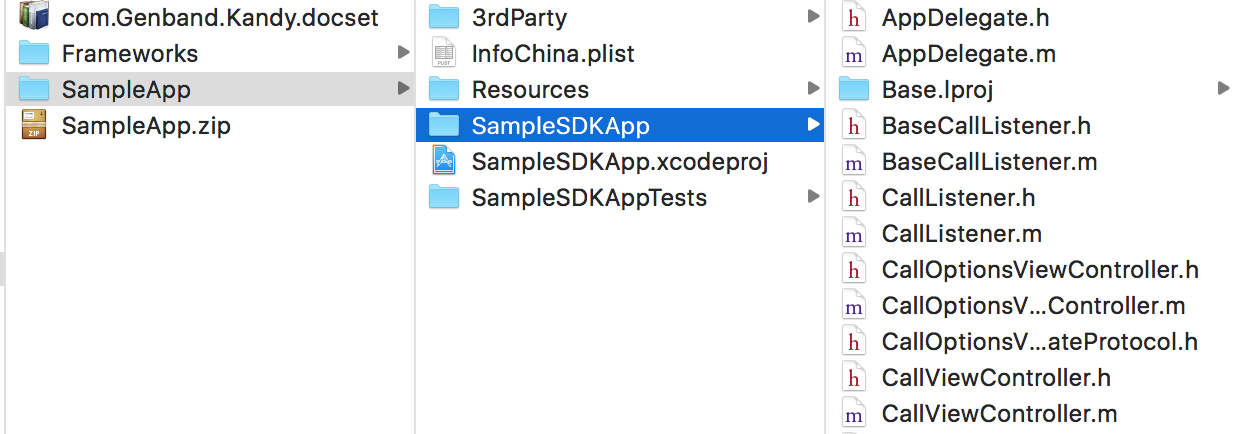
## IOS SDK下载

KANDY的IOS SDK是一个IOS库文件(KandySDK.framework)。这个库文件包含了呼叫、消息、群组、协同等功能。这个库文件放在Kandy的云端，有各种历史版本，1.6版本为例，其库文件的连接可见：

<https://developer.kandycn.com/code/ios-reference-application>



下载解压后的目录结构如下：



* com.Genband.kandy.docset ：kandysdk 的docset格式说明文档。
* Frameworks：是ios kandy sdk 库的目录
* SampleSDKApp：DEMO工程。

## 运行SampleSDKApp工程

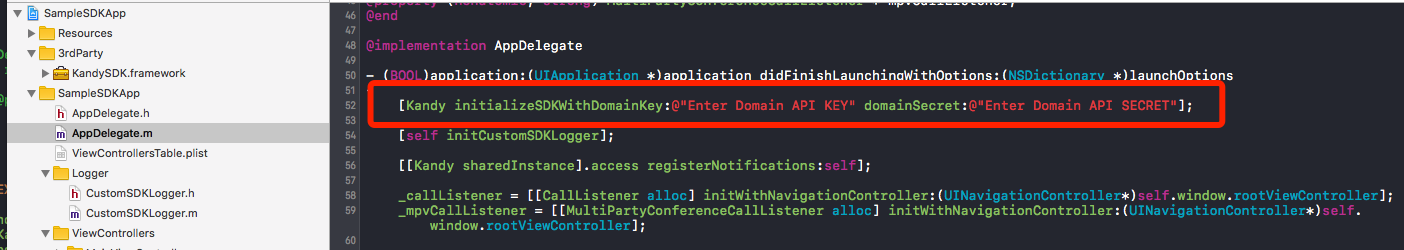
#### 步骤一: 前提

1.OS X 系统环境

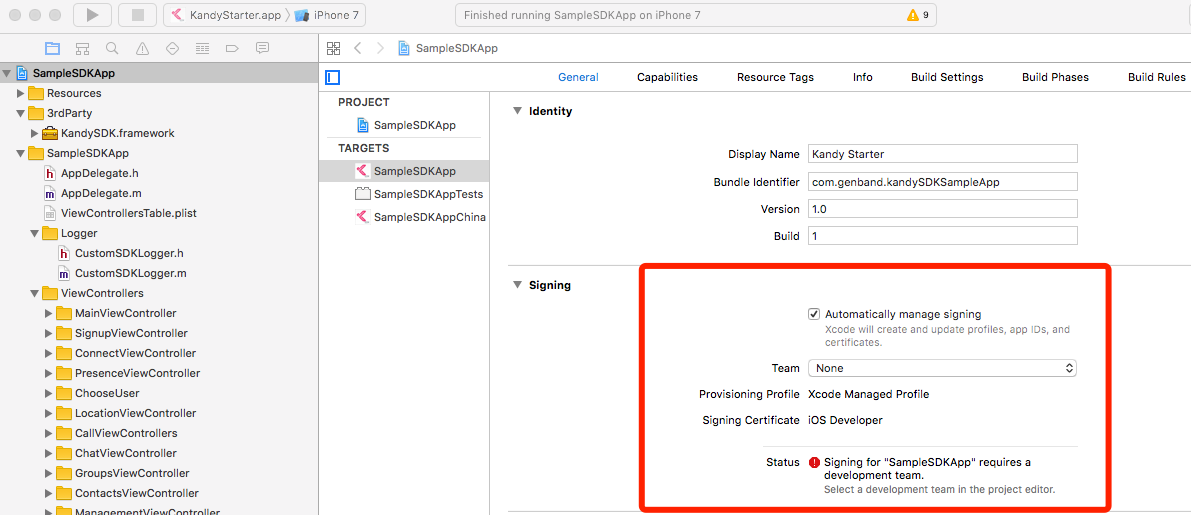
2.最新版本的Xcode 开发环境。

注意：Kandy SDK 支持到 ios7+。

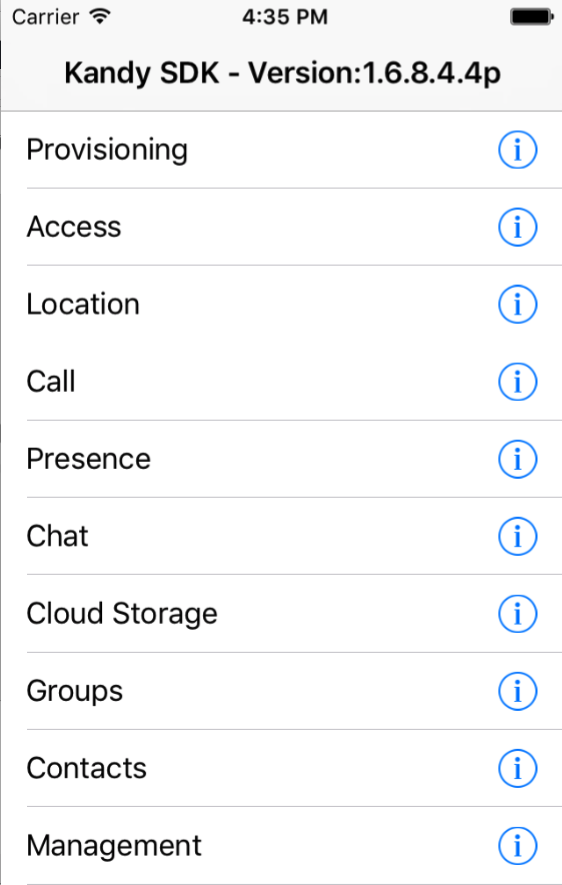
#### 步骤二：打开SampleSDKApp工程，并设置APP Key和 App Secret



#### 步骤三：设置真机运行证书



#### 步骤四：SampleSDKApp说明



Provisioning：用户注册

Access：用户登录和登出模块

Location：用户定位

Call：呼叫，单点通话和多方会议

Presence：用户在线状态

Chat：聊天，包含单聊和群聊

Cloud Storage：云存储

Groups：用户群

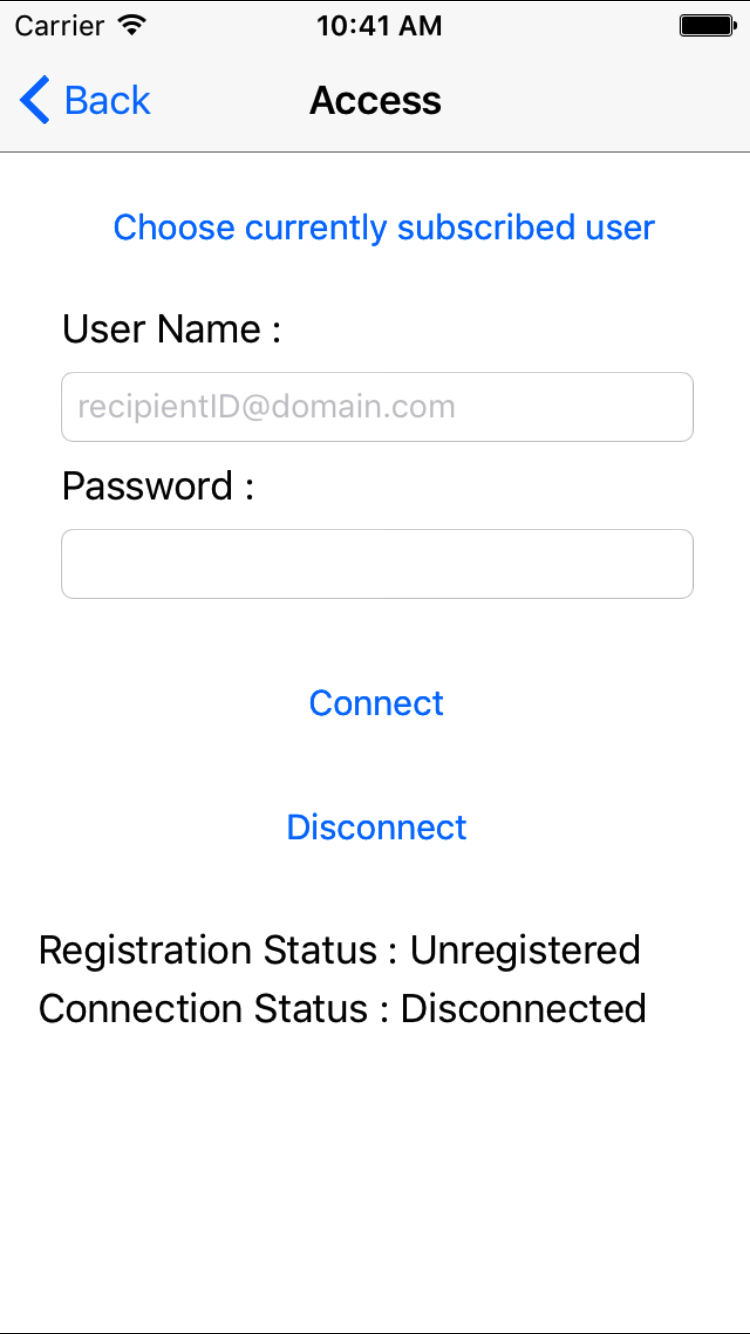
Contacts：通讯录，包括云端通讯录和本地通讯录

Management：管理相关，设置app key，收集log， 发送log

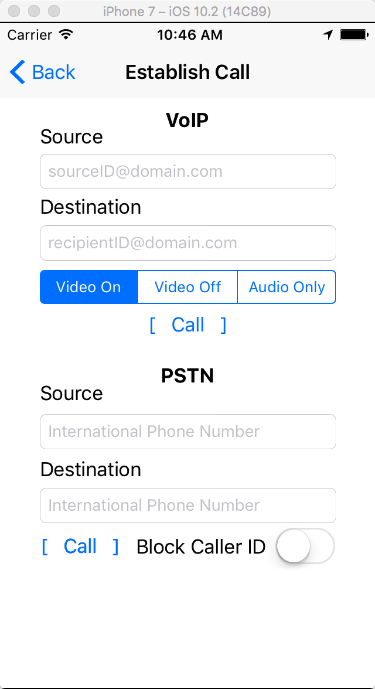
## SampleSDKApp拨打电话

在开始之前，您需要根据章节1.4的指导先创建您的账号，并得到您的至少2个用户。

#### 账户登录



#### 拨打电话



# IOS SDK使用

## 如何使用本文档

为了让您更快了解KANDY提供的功能，本文档将Kandy的基本功能归类并做出讲解。为了方便理解，基本每个功能我们都会做详细的讲解，并在Demo的工程中体现出来。请大家看本文前，务必完成KandySDK.framework [下载](https://developer.kandycn.com/code/ios-reference-application),，并在真机上成功运行SampleSDKApp，过程请参考章节2.2。作为对照学习，你将会更佳便捷地体验Kandy的精彩功能。

对于每个函数具体的功能说明，可能本文档中不能涵盖到，您可以去Kandy的API文档网站去查询具体的函数使用方法

所有IOS SDK的使用说明，可以参见：

<https://developer.kandycn.com/docs/ios/1-6>

您准备好了么，让我们开始吧！

## 开始之前

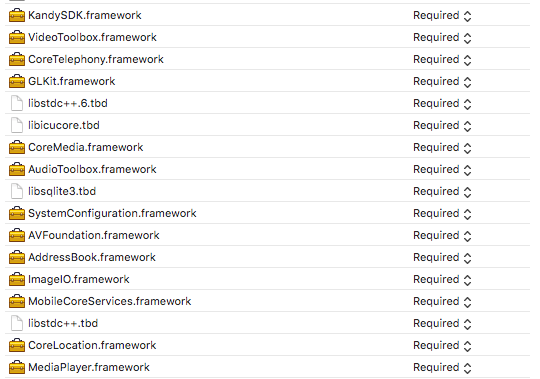
在开始之前您需要完成如新几个简单步骤：

* ***步骤一：创建账号***

在开始之前，您需要根据章节1.4的指导先创建您的账号，并得到您的至少2个用户。

* ***步骤二：***KandySDK.framework***引用***

为了使用Kandy的IOS SDK，您需要将上节的库文件包含入您自己的IOS程序源码中，并写需要将KandySDK.framework 依赖的第三方库也加入到工程。如图：



具体罗列：

VideoToolbox.framework

CoreTelephony.framework

GLKit.framework

libstdc++.6.dylib

libicucore.dylib

CoreMedia.framework

AudioToolbox.framework

libsqlite3.dylib

SystemConfiguration.framework

AVFoundation.framework

AddressBook.framework

ImageIO.framework

MobileCoreServices.framework

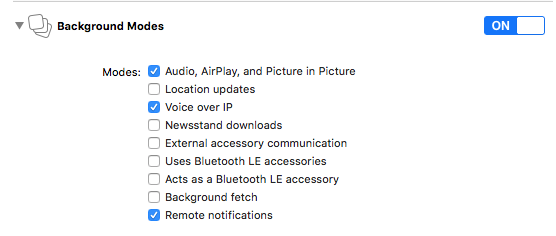
Libc++.dylib

CoreLocation.framework

MediaPlayer.framework

***步骤三：APP 工程配置***

KandySDK 是对Voip 功能实现的封装，故需要引入如下的后台运行模式：audio，voip和notification［如需要进入voip push］。



***步骤四：设置API Key 和hostUrl***

您需要知道所使用的用户所在的项目的API Key和Secret，这些信息在后面SDK的环境配置中需要输入。

-(BOOL)application:(UIApplication \*)application didFinishLaunchingWithOptions:(NSDictionary \*)launchOptions {

[Kandy initializeWithDomainKey:@"YourDomainKey" domainSecret:@"YourDomainSecret"];

[Kandy sharedInstance].globalSettings.kandyServiceHost = @"https://api.kandycn.com";

[Kandy sharedInstance].globalSettings.kandyServiceTimeout = 30;

[self initCustomSDKLogger];

return YES;

}

-(void)initCustomSDKLogger{

CustomSDKLogger \* customSDKLogger = [[CustomSDKLogger alloc] initWithFormatter:[Kandy sharedInstance].loggingInterface.loggingFormatter];

[Kandy sharedInstance].loggingInterface = customSDKLogger;

}

代码说明

* + [Kandy initializeWithDomainKey:@"YourDomainKey" domainSecret: @"YourDomainSecret"]; 设置ApiKey 和 ApiSecret
  + [Kandy sharedInstance].globalSettings.kandyServiceHost = @"https://api.kandycn.com"; 设置hostUrl 为kandycn域名［要设置］
  + [Kandy sharedInstance].globalSettings.kandyServiceTimeout = 30; 设置kandy的超时时间。
  + [self initCustomSDKLogger]; 设置kandy日志输出方式采用文件输出，不设置将在console 中输出。详情请参考Demo project AppDelegate.m文件。

到此我们已经完成kandySDK project 环境的配置全部工作。

代码参考：SampleApp/SampleSDKApp/AppDelegate.m

文档参考：<https://developer.kandycn.com/tutorials/ios/1-6#quick-start>.

下面让我们开始美妙的kandySDK 开发之旅。

## 用户注册

Kandy 支持两种方式来完成用户注册

1，Admin账号可以在kandycn管理台注册用户，参考章节1.4。

2，用户可以通过SDK调用来完成注册，当前Kandy sdk 版本已经支持通过手机号认证来完成注册过程。

通过手机号完成用户步骤如下：

##### 步骤一：用户请求激活码

支持通过短信和IVR的方式发送激活码，用户输入收到的激活码进入步骤二。

-(void)registerSubscriber{

KandyAreaCode \* areaCode = [[KandyAreaCode alloc] initWithISOCode:@"cn " andCountryName:@"China" andPhonePrefix:@"+86"];

NSString \* strPhoneNumber = @"<phone number>";

[[Kandy sharedInstance].provisioning requestCode:areaCode phoneNumber:strPhoneNumber codeRetrivalMethod:EKandyValidationMethod\_sms responseCallback:^(NSError \*error, NSString \*destinationToValidate) {

if (error) { // Failure

} else { // Success

}

}];

}

IVR是通过给用户回拨电话方式，发送激活码。注意：假如用户没有应答或则拒绝回拨电话，那个这个函数调用将返回Error。

##### 步骤二：验证验证码

SDK将发送收到的激活码到Kandy 服务器。一旦验证通过，系统将创建这个新的用户，并将用户信息返回到callback中。

-(void)validate{

NSString \* validationCode = @"9876";

KandyAreaCode \* areaCode = [[KandyAreaCode alloc] initWithISOCode:@"cn" andCountryName:@"china" andPhonePrefix:@"+86"];

NSString \* strPhoneNumber = @"180000000000";

[[Kandy sharedInstance].provisioning validateAndProvision:validationCode destination:strPhoneNumber areaCode:areaCode responseCallback:^(NSError \*error, KandyUserInfo \*provisionedUserInfo) {

if (error) { // Failure

} else { // Success

}

}];

}

代码参考：SampleApp/SampleSDKApp/ SignupViewController.m

文档参考：

<https://developer.kandycn.com/tutorials/ios/1-6#subscriber-provisioning>

## 用户登录

在完成用户注册后，开始登录系统操作。这个是在使用Kandy 通讯工作前的最后一步。

#### 登录Delegates

登录Delegates 是在用户的登录状态发生改变时，调用Delegates函数。

实现 KandyAccessNotificationDelegate函数在子类中。

注册登录delegate，代码如下：

-(void)registerForAccessEvents{

[[Kandy sharedInstance].access registerNotifications:self];

}

完成登录delegate注册后，实现如下函数：

-(void) registrationStatusChanged:(EKandyRegistrationState)registrationStatus{ // Handle registration change

}

-(void) connectionStatusChanged:(EKandyConnectionState)connectionStatus{ // Handle connection change

}

-(void) gotInvalidUser:(NSError\*)error{

// Invalid user

}

-(void) sessionExpired:(NSError\*)error{

// Session expired, you can renew it by calling to following method

[[Kandy sharedInstance].access renewExpiredSession:^(NSError \*error) {

}];

}

-(void) SDKNotSupported:(NSError\*)error{

// Failure

}

#### 后台运行模式

KandySDK要求在应用进入后台，其能维持通话状态。

故我们需要设置在应用后台时候设置维持链接处于活动状态。

- (void)applicationWillResignActive:(UIApplication \*)application {

// Sent when the application is about to move from active to inactive state. This can occur for certain types of temporary interruptions (such as an incoming phone call or SMS message) or when the user quits the application and it begins the transition to the background state.

// Use this method to pause ongoing tasks, disable timers, and throttle down OpenGL ES frame rates. Games should use this method to pause the game.

[[UIApplication sharedApplication] setKeepAliveTimeout:600 handler:^{

[[Kandy sharedInstance].access keepAliveTimeoutHandler];

}];

}

#### 登录

采用用户名和密码的登录方式的代码如下：

-(void)login {

KandyUserInfo \* userInfo = [[KandyUserInfo alloc] initWithUserId:@"user1@domain" password:@"UserPassword"];

[[Kandy sharedInstance].access login:userInfo responseCallback:^(NSError \*error) {

if (error) { // Failure

} else { // Success }

}];

}

#### 更新过期会话

会话过期是时常发送的事情。所以当登录delegate 出现会话过期是，如下代码达到更新会话的目的。

[[Kandy sharedInstance].access renewExpiredSession: :^(NSError \*error) {

if (error) { // Failure

} else { // Success

}

}];

#### 登出

登出当前的会话状态。

-(void)logout{

[[Kandy sharedInstance].access logoutWithResponseCallback:^(NSError \*error) {

if (error) { // Failure

} else { // Success }

}];

}

#### Connection和registration 状态

主动获取connection和registration在服务器的状态，可以使用如下代码：

[[Kandy sharedInstance].access.connectionState

[[Kandy sharedInstance].access.registrationState

Connection 状态描述当前用户的登录状态

Registration 状态藐视当前用户的注册状态

代码参考：SampleApp/SampleSDKApp/ConnectViewController.m

文档参考：

<https://developer.kandycn.com/tutorials/ios/1-6#access>

## 推送

推送服务提供能力去接受和处理推送消息。

### 前提条件

在开启推送前，请完成如下的步骤（如下的支持在ios8+的系统中运行）。

更多的信息请访问apple 官方文档，[Local and Remote 推送编程指南](https://developer.apple.com/library/prerelease/content/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/)。

1. 在[ios developer](https://developer.apple.com/account/ios/profile/) 中创建Voip push 证书，将倒出证书为\*.pem格式，具体步骤请参考[网站教程](https://blog.serverdensity.com/how-to-renew-your-apple-push-notification-push-ssl-certificate/)。
2. 将其以邮件附件的形式并附带App的Bundle Id和Domain一起发送到 [support@kandy.io](mailto:support@kandy.io) 邮箱中。
3. 在appDelegate中，设置开启通知。

-(BOOL)application:(UIApplication \*)application didFinishLaunchingWithOptions:(NSDictionary \*)launchOptions {

//initialize the KandySDK

if ([[UIApplication sharedApplication] respondsToSelector:@selector(registerUserNotificationSettings:)]) {

//Register UserNotificationSettings

[[UIApplication sharedApplication] registerUserNotificationSettings:[UIUserNotificationSettings settingsForTypes:(UIUserNotificationTypeBadge|UIUserNotificationTypeSound|UIUserNotificationTypeAlert) categories:nil]];

} else {

[[UIApplication sharedApplication] registerForRemoteNotificationTypes:(UIRemoteNotificationType)(UIRemoteNotificationTypeBadge|UIRemoteNotificationTypeSound|UIRemoteNotificationTypeAlert)];

}

}

1. 实现获取用户通知权限，开启注册remote的通知。

-(void)application:(UIApplication \*)application

didRegisterUserNotificationSettings:(UIUserNotificationSettings \*)notificationSettings {

[[UIApplication sharedApplication] registerForRemoteNotifications];

}

1. 存放收到的deviceToken，为了后面使用。

-(void)application:(UIApplication \*)application didRegisterForRemoteNotificationsWithDeviceToken:(NSData \*)deviceToken {

[[NSUserDefaults standardUserDefaults]setObject:deviceToken forKey:@"deviceToken"];

NSString\* bundleId = [[NSBundle mainBundle]bundleIdentifier];

}

### Voip推送

只从ios8.1开始，IOS有个新的推送方式－Voip推送。他提供更多的额外的功能，比如：当app 在后台运行，可以提供CPU调度资源让app处理推送事件后通知到用户。[在apple 官方网站上](https://developer.apple.com/reference/pushkit)提供更多的关于Voip 推送和PushKit Framework的资料。为注册Voip推送，我们需要完成如下代码，将标准推送替换为Voip推送。如下的代码是使用PushKit简单样例。

1. 创建class 实现PKPushRegistryDelegate协议，初始化PKPushRegistry对象。

PKPushRegistry \* pushRegistry = [[PKPushRegistry alloc] initWithQueue:dispatch\_get\_global\_queue(DISPATCH\_QUEUE\_PRIORITY\_DEFAULT, 0)];

pushRegistry.delegate = self;

pushRegistry.desiredPushTypes = [NSSet setWithObject:PKPushTypeVoIP];

1. 实现PKPushRegistryDelegate

-(void)pushRegistry:(PKPushRegistry \*)registry didUpdatePushCredentials:(PKPushCredentials \\*)credentials forType:(NSString \*)type{

// Registered for VoIP push notifications. Store push token for future use

[[NSUserDefaults standardUserDefaults]setObject:credentials.token forKey:@voipToken];

}

-(void)pushRegistry:(PKPushRegistry \*)registry didReceiveIncomingPushWithPayload:(PKPushPayload \*)payload forType:(NSString \*)type{

// Received VoIP Push Notification, handle it

}

-(void)pushRegistry:(PKPushRegistry \*)registry didInvalidatePushTokenForType:(NSString \*)type {

// VoIP Push credentials invalidated

NSString\* bundleId = [[NSBundle mainBundle]bundleIdentifier]

[Kandy sharedInstance].services.push disableRemoteNotificationsWithBundleId: bundleId responseCallback:^(NSErro \*error) {

if (error) {

// Handle error

}

}];

}

### 开启推送功能

在完成准备上述的准备步骤后，在app启动中我们就能处理推送消息。

在登录完成后，通过如下代码中将token 传到kandy中，这样kandy就具备发送推送消息给你的能力。

[[Kandy sharedInstance].access login:kandyUserInfo andResponseCallback:^(NSError \*error) {

if (!error) {

NSData\* deviceToken = [[NSUserDefaults standardUserDefaults]objectForKey: isVoip ? @"voipToken" : @"deviceToken"];

NSString\* bundleId = [[NSBundle mainBundle]bundleIdentifier];

[[Kandy sharedInstance].services.push enableRemoteNotificationsWithToken:deviceToken bundleId:bundleId isSandbox:isSandbox isVoipPush:isVoip responseCallback:^(NSError \*error) {

if (error) {

//handle error e.g no Internet connection

}

}];

}

}];

enableRemoteNotificationsWithToken是上送deviceToken到kandy 云端，它可以更新设备的推送deviceToken。

### 处理推送

在用户完成登录后，我们开始处理kandy 推送消息。

App启动时候处理Kandy SDK推送。

-(BOOL)application:(UIApplication \*)application didFinishLaunchingWithOptions:(NSDictionary \*)launchOptions {

//initialize the KandySDK //register user notification settings

if (launchOptions && [launchOptions objectForKey:UIApplicationLaunchOptionsRemoteNotificationKey] != nil) {

NSDictionary\* remoteNotification = [launchOptions objectForKey:UIApplicationLaunchOptionsRemoteNotificationKey];

id<KandyEventProtocol> event = [[Kandy sharedInstance].notifications getRemoteNotificationEvent:remoteNotification];

[self\_handleEventUI:event];

// event is nil if not handled by Kandy

[[Kandy sharedInstance].notifications handleRemoteNotification:remoteNotification responseCallback: ^(NSError \*error){

if (error && error.domain isEqualToString:kandyNotificationServiceErrorDomian) && error.code == EKandyNotificationServiceError\_pushFormatNotSupported) { // Could not handle remote notification

} else {

// Success

}

}};

}

}

当app处于运行状态时，收到kandy sdk 推送消息可以立刻处理。

-(void)application:(UIApplication \*)application didReceiveRemoteNotification:(NSDictionary \*)userInfo {

[[Kandy sharedInstance].services.push handleRemoteNotification:userInfo responseCallback:^(NSError \*error) {

if (error && [error.domain isEqualToString:KandyNotificationServiceErrorDomain] && error.code == EKandyNotificationServiceError\_pushFormatNotSupporte) {

//Push format not supported by Kandy, handle the notification my self }

}];

}

如下代码可以让通知消息转化为事件处于链接状态，这样也许有助于做UI更新。

id<KandyEventProtocol> event = [[Kandy sharedInstance].services.push getRemoteNotificationEvent:remoteNotification];

代码参考：SampleApp/SampleSDKApp/ AppDelegate.m

文档参考：

<https://developer.kandycn.com/tutorials/ios/1-6#push-notifications>

## 定位

### 国家信息

依据当前的位置［SIM或IP地址］，获取当前的国家代码。

基于位置获取国家代码，代码如下：

-(void)getCountryInfo {

[[Kandy sharedInstance].services.location getCountryInfoWithResponseCallback:^(NSError \*error, KandyAreaCode \*areaCode) {

if (error) { // Failure

} else { // Success

}

}];

}

代码参考：SampleApp/SampleSDKApp/LocationViewController.m

文档参考：

<https://developer.kandycn.com/tutorials/ios/1-6#location>

## 呼叫

在本章节中，将讲解呼叫设置，发起Voip 和PSTN 呼叫及处理常见的呼叫形式。在每个子章节结束，都提供有效的事例代码。

### 呼叫设置

在app使用任何呼叫的特性前，需要了解呼叫的相关设置。KandyCallServiceSettings是负责呼叫设置的相关主类。如下我们罗列呼叫设置函数。

1. defaultCameraPosition，摄像头位置，在视频通话开始活着过程中我们可以对其调制，默认值：EKandyCameraPosition\_front
2. cameraOrientationSupport，摄像头方向，在视频通话开始活着过程中我们可以对其调制，默认值：EKandyCameraOrientationSupport\_statusBar
3. shouldRejectIncomingCallsWhileInGSMCall，在GSM通话过程中，新的呼叫是否被自动拒绝，默认值：YES。

### 发起Voip呼叫

发布一个对外的呼叫，调用createVoipCall: callee: options:。其参数说明如下：

callee，[被叫的完整用户，名如user@example.kandy.io](mailto:被叫的完整用户名如user@example.kandy.io)

caller，主叫的用户名，是选填项。

isStartCallWithVideo，是Boolean参数，是否caller发起视频呼叫。

渲染视频流的容器

localView，UIKit view的对象，绑定当前view对象到localView，将显示本地摄像头采样数据。

remoteView，UIKit view的对象，绑定当前view对象到remoteView，将显示远端流数据。

-(void) makeVoipCall {

KandyRecord \* kandyRecord = [[KandyRecord alloc] initWithURI:<callee@s username> type:EKandyRecordType\_contact];

KandyRecord \*initiator = [[KandyRecord alloc] initWithURI:<Callers@full username> type:EKandyRecordType\_contact];

EKandyOutgingVoIPCallOptions outgoingCallOption = EKandyOutgingVoIPCallOptions\_startCallWithVideo;

[[Kandy sharedInstance].services.call createVoipCall:initiator callee:kandyRecord options:outgoingCallOption responseCallback:^(NSError \* \_Nullable error, id<KandyOutgoingCallProtocol> \_Nullable kandyOutgoingCall) {

\_\_block CallViewController \* vcCall = [self.storyboard instantiateViewControllerWithIdentifier:NSStringFromClass([CallViewController class])];

typeof(self) \_\_weak weakSelf = self;

[kandyOutgoingCall establishWithResponseBlock:^(NSError \*error) {

dispatch\_async(dispatch\_get\_main\_queue(), ^{

if (!error) { [weakSelf.navigationController presentViewController:vcCall animated:YES completion:^{}];

}

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

message:error.localizedDescription

delegate:weakSelf cancelButtonTitle:@"OK" otherButtonTitles:nil];

[alert show];

}

});

}];

}];

### 发起Voip到PSTN呼叫

如下的函数初始化Voip到PSTN呼叫。

International phone number 是目标手机号码

注意：目标手机号必须符合E164规范，不含＋前缀。如：US电话［213］456-7890，电话是：12134567890

-(void) makePSTNCall {

[[Kandy sharedInstance].services.call createPSTNCall:<> destination:self.txtPstnDestination.text options:self.blockedCallerIDSwitch.isOn ? EKandyOutgingPSTNCallOptions\_blockedCallerID : EKandyOutgingPSTNCallOptions\_none responseCallback:^(NSError \* \_Nullable error, id<KandyOutgoingCallProtocol> \_Nullable kandyOutgoingCall) {

\_\_block CallViewController \* vcCall = [self.storyboard instantiateViewControllerWithIdentifier:NSStringFromClass([CallViewController class])];

typeof(self) \_\_weak weakSelf = self;

[kandyOutgoingCall establishWithResponseBlock:^(NSError \*error) { dispatch\_async(dispatch\_get\_main\_queue(), ^{ if (!error) {

[weakSelf saveGuiValues]; [weakSelf.navigationController presentViewController:vcCall animated:YES completion:^{}];

} else {

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

message:error.localizedDescription delegate:weakSelf cancelButtonTitle:@"OK"

otherButtonTitles:nil];

[alert show];

}

});

}];

### 呼叫状态监听

注册KandyCallServiceNotificationDelegate代理，将获取incomming call、callStateChange、MissCall 等等通知。

添加KandyCallServiceNotificationDelegate代理函数

[[Kandy sharedInstance].services.call registerServiceNotificationsDelegate:self];

取消KandyCallServiceNotificationDelegate代理函数

[[Kandy sharedInstance].services.call unregisterNotifications:self];

KandyCallServiceNotificationDelegate代理说明

1. gotIncomingCall :(id<KandyIncomingCallProtocol>)call

当有incomming call进入，此函数会被调用。通知用户coming call，并且实现接听和拒绝的操作。

[[Kandy sharedInstance].services.call unregisterNotifications:self];

-(void) gotIncomingCall:(id<KandyIncomingCallProtocol>)call

{

BOOL isAnswerWithVideo = YES;

[call accept:isAnswerWithVideo withResponseBlock:^(NSError \*error) {

if(error){

//Failure

}

else

{

//Success

}

}];

}

1. gotMissedCall :(KandyMissedCall\*)missedCall

当我有个missed call，该函数将会被调用。

-(void) gotMissedCall:(KandyMissedCall\*)missedCall{

}

1. stateChanged:forCall:

将监听callStatusChange，如Call initiated, call connected, talking等。

-(void) stateChanged:(EKandyCallState)callState forCall:(id<KandyCallProtocol>)call{

if (callState == EKandyCallState\_unknown) {

// code goes here

}

if (callState == EKandyCallState\_initialized) { // code goes here

}

if (callState == EKandyCallState\_dialing) {

// code goes here

}

if (callState == EKandyCallState\_sessionProgress) {

// code goes here

}

if (callState == EKandyCallState\_ringing) {

// code goes here

}

if (callState == EKandyCallState\_answering) {

// code goes here

}

if (callState == EKandyCallState\_talking) {

// code goes here

}

if (callState == EKandyCallState\_terminated) {

// code goes here

}

if (callState == EKandyCallState\_notificationWaiting) {

// code goes here

}

}

1. availableAudioOutputChanged

audio 输出状态改变的通知函数

-(void) availableAudioOutputChanged:(NSArray\*) updatedAvailableAudioOutputs {

}

### 呼叫应答操作

kandySDK提供对当前有效的呼叫多种操作，如mute, switch camera, hold…。为此，我们将学习相关的操作。

##### Holding a call 保持呼叫

当call进来，用户可以hold 或unhold 来电。

* + - * (IBAction)didTapHold:(id)sender {
      * if (self.kandyCall.isOnHold) {
      * [self.kandyCall unHoldWithResponseCallback:^(NSError \*error) {
      * [self updateButtonStates];
      * if (error) {
      * UIAlertView \*alert = [[UIAlertView alloc] initWithTitle:@"Error" message:error.localizedDescription delegate:self cancelButtonTitle:@"OK" otherButtonTitles:nil]; [alert show]; } }]; } else {
      * [self.kandyCall holdWithResponseCallback:^(NSError \*error) {
      * [self updateButtonStates];
      * if (error) {
      * UIAlertView \*alert = [[UIAlertView alloc] initWithTitle:@"Error" message:error.localizedDescription delegate:self
      * cancelButtonTitle:@"OK" otherButtonTitles:nil]; [alert show]; } }]; } }

##### ending a call 结束呼叫

结束呼叫，需要调用kandy hangupWithResponseCallback，结束激活呼叫。

* + - * (void) hangupCall:(NSString \*)callURI {
      * id<KandyCallProtocol>call = [[Kandy sharedInstance].services.call getCallByRemoteRecord:[[KandyRecord alloc] initWithURI:callURI]];
      * typeof(self) \_\_weak weakSelf = self;
      * [call hangupWithResponseCallback:^(NSError \*error){
      * dispatch\_async(dispatch\_get\_main\_queue(), ^{
      * if (error) {
      * UIAlertView \*alert = [[UIAlertView alloc] initWithTitle:@"Error" message:error.localizedDescription delegate:weakSelf cancelButtonTitle:@"OK" otherButtonTitles:nil];
      * [alert show];
      * } else if ([weakSelf.activeCall.remoteRecord.uri isEqualToString:callURI])
      * { weakSelf.activeCall = nil; [weakSelf setupGui];
      * }
      * });
      * }];
      * }

##### muting a call 静音呼叫

静音呼叫，调用muteWithResponseCallback 和 unmuteWithResponseCallback 函数。静音呼叫会停止语音负载传输。

* + - * －(IBAction)didTapMute:(id)sender {
      * if (self.kandyCall.isMute) {
      * [self.kandyCall unmuteWithResponseCallback:^(NSError \*error) {
      * [self updateButtonStates];
      * if (error) {
      * UIAlertView \*alert = [[UIAlertView alloc] initWithTitle:@"Error"
      * message:error.localizedDescription
      * delegate:self cancelButtonTitle:@"OK" otherButtonTitles:nil]; [alert show];
      * }
      * }];
      * } else {
      * [self.kandyCall muteWithResponseCallback:^(NSError \*error) {
      * [self updateButtonStates];
      * if (error) { UIAlertView \*alert = [[UIAlertView alloc] initWithTitle:@"Error"
      * message:error.localizedDescription delegate:self cancelButtonTitle:@"OK"
      * otherButtonTitles:nil];
      * [alert show];
      * }
      * }];
      * } }

##### changing video 状态

在视频通话中可开始和停止视频负载的传输，调用startVideoSharingWithResponseCallback和stopVideoSharingWithResponseCallback函数。

* + - * (IBAction)didTapVideoOnOff:(id)sender {
      * if (self.kandyCall.isSendingVideo) {
      * [self.kandyCall stopVideoSharingWithResponseCallback:^(NSError \*error) {
      * [self updateButtonStates];
      * if (error) { UIAlertView \*alert = [[UIAlertView alloc] initWithTitle:@"Error"
      * message:error.localizedDescription
      * delegate:self cancelButtonTitle:@"OK" otherButtonTitles:nil]; [alert show];
      * }
      * }];
      * } else {
      * [self.kandyCall startVideoSharingWithResponseCallback:^(NSError \*error) {
      * [self updateButtonStates];
      * if (error) {
      * UIAlertView \*alert = [[UIAlertView alloc] initWithTitle:@"Error"
      * message:error.localizedDescription delegate:self
      * cancelButtonTitle:@"OK"

* + - * otherButtonTitles:nil];
      * [alert show];
      * }
      * }];
      * }
      * }

### 呼叫统计

KandyCallProtocol 可以倒出链接的质量信息。

调用获取呼叫统计函数如下：

id<KandyCallProtocol> call = <Call object>;

[call getRTPStatisticsWithCompletion:^(id<KandyCallRTPStatisticsProtocol> statistics, NSError \*error) {

// Handle statistics information

}]

代码参考：SampleApp/SampleSDKApp/CallViewController

文档参考：

<https://developer.kandycn.com/tutorials/ios/1-6#call>

## 多人会议

多人会议让你创建视频/语音会议房间，后邀请别人加入你的会议。

### 加入房间

在上面的步骤完成，我们对房间发起呼叫让后通知服务器加入会议室。如下是加入会议的代码，参数说明：

(NSString \*) conferenceID 会议ID你收到在你创建会议房间。

(NSString \*) nickname 在房间中设置你的昵称

/\*\*

\* Join a multi party conference

\*

\* @param conferenceID The conference ID

\* @param nickName The name that will be shown in room

\*/

-(void)join:(NSString \*)conferenceID nickName:(NSString \*)nickName { [[Kandy sharedInstance].services.multiPartyConference join:conferenceID nickName:nickName responseCallback:^(NSError \*error) {

if (nil != error) {

// handle error

} else {

// handle response

}

}];

}

### 离开房间

一旦离开会议，需要通知服务器离开会议室。离开会议代码如下：

函数参数：

(NSString \*) conferenceID 会议ID你收到在你创建会议房间。

/\*\*

\* Leave a multi party conference \*

\* @param conferenceID The conference ID

\*/

-(void)leave:(NSString \*)conferenceID responseCallback:(void(^)(NSError \* error))responseCallback {

[[Kandy sharedInstance].services.multiPartyConference leave:\_conferenceRoomDetails.conferenceID responseCallback:^(NSError \*error) {

if (nil != error) { // handle error

} else {

// handle response } }];

}

### 销毁房间

在完成会话和全部参会者都离开会议室，需要将房间销毁。销毁的房间的函数如下，参数说明：

(NSString \*) conferenceID 会议ID你收到在你创建会议房间。

/\*\*

\* Destroy a multi party conference room

\*

\* @param conferenceID The conference ID

\*/

-(void)destroyRoom:(NSString \*)conferenceID {

[[Kandy sharedInstance].services.multiPartyConference destroyRoom:(NSString \*)conferenceID responseCallback:^(NSError \* error) { if (nil != error) {

// handle error }

else { // handle response }

}];

}

### 获取会议状态

获取激活的会议呼叫当前的状态。

参数说明：

(NSString \*) conferenceID 会议ID你收到在你创建会议房间。

/\*\*

\* Get an active conference call info \*

\* @param conferenceID The conference ID

\*/

- (void) getConferenceStatus:(NSString \*)conferenceID { [[Kandy sharedInstance].services.multiPartyConference getConferenceCallDetailsWithConferenceID:conferenceID responseCallback:^(NSError \*error, id<KandyMultiPartyConferenceCallDetailsProtocol> conferenceCallDetails) {

if (nil != error) {

// handle error } else {

// handle response } }];

}

### 更新参会者姓名

更新激活会议电话的参会者姓名，该权限仅仅对会议的adminstrater 有效。

参数说明：

(NSString \*) newNickName新的昵称。

(NSString \*) participantID 更新参会者ID。

(NSString \*) conferenceID 会议ID你收到在你创建会议房间。

/\*\*

\* Update a participant name \*

\* @param newNickname The new name

\* @param participantID The participant ID

\* @param conferenceID The conference ID

\*/

- (void) updateParticipantName:(NSString \*)newNickname participantID:(NSString \*)participantID conferenceID:(NSString \*)conferenceID {

[[Kandy sharedInstance].services.multiPartyConference updateParticipantName:newNickname participantID:participantID conferenceID:conferenceID responseCallback:^(NSError \*error) {

if (nil != error) {

// handle error } else { // handle response

}

}];

}

### 会议控制

对其他参会者的操作，该权限仅仅对会议的adminstrater 有效。

支持操作：

Mute

Unmute

Hold

Unhold

Enable Video

Disable Video

Remove from Conference

参数说明：

(EKandyMultiPartyConferenceAction\*) action 操作动作标示。

(NSString \*) participantID 更新参会者ID。

(NSString \*) conferenceID 会议ID你收到在你创建会议房间。

/\*\* \* Update participant available actions \*

\* @param participantID The participant ID

\* @param action The control action

\* @param conferenceID The conference ID

\*/

- (void) updateContorlActionForParticipantWithID:(NSString \*)participantID action:(EKandyMultiPartyConferenceAction)action conferenceID:(NSString \*)conferenceID {

KandyMultiPartyConferenceParticipantActionParams \* participantActions = [[KandyMultiPartyConferenceParticipantActionParams alloc] initWithParticipantID:participantID action:action]; [[Kandy sharedInstance].services.multiPartyConference updateRoomParticipantActions:@[participantActions] conferenceID:conferenceID responseCallback:^(NSError \* error) { if (nil != error) { // handle error } else {

// handle response }

}]; }

## 聊天

chat Service 支持单人和群发送／接受文本和富媒体消息。

确保你已开启推送功能，在应用处于非激活状态时将收到新的消息和🕐推送。

### 聊天状态监听

通过监听可以接受聊天相关的通知(onChatReceived, onChatDelivered, onAutoDownloadProgress, onAutoDownloadFinished).

为此，需要实现 KandyChatServiceNotificationDelegate 函数。

如希望多次收到消息，则不用发送确认收到聊天事件。

代码如下：

-(void)registerForChatEvents {

[[Kandy sharedInstance].services.chat registerNotifications:self];

}

处理chat事件，代码如下：

-(void)onMessageReceived:(id<KandyMessageProtocol>)kandyMessage recipientType:(EKandyRecordType)recipientType{

switch (kandyMessage.mediaItem.mediaType) {

case EKandyFileType\_text: //Your code here break;

case EKandyFileType\_image: //Your code here break;

case EKandyFileType\_video: //Your code here break;

case EKandyFileType\_audio: //Your code here break;

case EKandyFileType\_location: //Your code here break;

case EKandyFileType\_contact: //Your code here break;

case EKandyFileType\_file: //Your code here break;

case EKandyFileType\_custom: //Your code here break;

default: break;

}

}

聊天消息的投递

-(void)onMessageDelivered:(KandyDeliveryAck \*)ackData {

//Your code here

}

自动下载

-(void) onAutoDownloadProgress:(KandyTransferProgress\*)transferProgress kandyMessage:(id<KandyMessageProtocol>)kandyMessage {

//Progress

}

自动下载完成

-(void) onAutoDownloadFinished:(NSError\*)error fileAbsolutePath:(NSString\*)path kandyMessage:(id<KandyMessageProtocol>)kandyMessage {

if(error){

//Failure

} else{

//Success

}

}

### 发送文本消息

发送文本消息，代码如下：

-(void)sendMsg {

KandyRecord \* kandyRecord = [[KandyRecord alloc] initWithURI:@"DestinationURI"];

KandyChatMessage \*textMessage = [[KandyChatMessage alloc] initWithText:@"Message text" recipient:kandyRecord];

[[Kandy sharedInstance].services.chat sendChat:chatMessage progressCallback:^(KandyTransferProgress \*transferProgress) { //progress }

responseCallback:^(NSError \*error) {

if (error) { //Failure

} else { //Success }

}];

}

### 发送组消息

发送群组消息，代码如下：

-(void)sendGroupMsg{

// if you want to create a group record

// KandyRecord \* kandyRecord = [[KandyRecord alloc] initWithUserName:@"groupUserName" domain:@"domain" type:EKandyRecordType\_group];

// if you already have a groupID

KandyRecord \* kandyRecord = self.kandyGroup.groupId;

KandyChatMessage \*textMessage = [[KandyChatMessage alloc] initWithText:@Message te” recipient:kandyRecord];

[[Kandy sharedInstance].services.chat sendChat:chatMessage progressCallback:^(KandyTransferProgress \*transferProgress){ //progress }

responseCallback:^(NSError \*error) {

if (error){ //Failure

} else { //Success }

}];

}

### 发送富媒体消息

发送图片消息，代码如下：

-(void)sendImageMsg {

KandyRecord \* kandyRecord = [[KandyRecord alloc] initWithURI:@"DestinationURI"];

id<KandyMediaItemProtocol> mediaItem = [KandyMessageBuilder createImageItem:@"absolutePathToImageFile text:@"Optional text"];

KandyChatMessage \*message = [[KandyChatMessage alloc] initWithMediaItem:mediaItem recipient:kandyRecord];

[[Kandy sharedInstance].services.chat sendChat:chatMessage progressCallback:^(KandyTransferProgress \*transferProgress){ //progress }

responseCallback:^(NSError \*error) {

if (error) { //Failure

} else { //Success }

}];

}

对于其他类型的富媒体消息，实现代码类似，只是在创建媒体类型上有所不同。

##### 视频

id<KandyMediaItemProtocol> mediaItem = [KandyMessageBuilder createVideoItem:@"absolutePathToImageFile" text:@"Optional text"];

##### 语音

id<KandyMediaItemProtocol> mediaItem = [KandyMessageBuilder createAudioItem:@"absolutePathToImageFile" text:@"Optional text"];

##### 位置

CLLocation \*location = [[CLLocation alloc] initWithLatitude:40.8283018 longitude:16.5500004];

id<KandyMediaItemProtocol> mediaItem = [KandyMessageBuilder createLocationItem:location text:@"Optional text"];

##### 联系人

[[Kandy sharedInstance].services.contacts createVCardDataByContact:kandyContact completionBlock:^(NSError \*error, NSData \*vCardData) { if (!error) {

NSString \*vcardPath = [NSTemporaryDirectory() stringByAppendingPathComponent:@"vcard.vcf"];

[vCardData writeToFile:vcardPath atomically:YES];

id<KandyMediaItemProtocol> contactMediaItem = [KandyMessageBuilder createContactItem:vcardPath text:@"Optional text];

}

}];

### 发送输入状态事件

chat service 支持向聊天对方［单人／群组］发送正在输入状态事件，但是chat service 本身不对其做任何处理。

发送正在输入状态事件，代码如下：

// Create typing indication object with start typing state

KandyTypingIndication \*typingIndication = [[KandyTypingIndication alloc] initWithState:EKandyUserTypingIndicationState\_typing contentType:@"text"];

// Send typing indication object to other user

[[Kandy sharedInstance].services.chat sendUserTypingIndication:typingIndication destination:otherUserRecord responseCallback:^(NSError \*error) {

if (error) {

LogError("Failed to send typing indication: %@ to user: %@", typingIndication, otherUserRecord);

} else {

LogInfo("Typing indication sent to %@: %@", otherUserRecord, typingIndication); }

}];

收到正在输入状态事件回调函数，如下：

-(void)onUserTypingIndicationReceived:(KandyTypingIndication \*)typingIndication sender:(KandyRecord \*)sender destination:(KandyRecord \*)destination {

switch (typingIndication.state) {

case EKandyUserTypingIndicationState\_typing: LogInfo("User %@ started typing!", sender);

break;

case EKandyUserTypingIndicationState\_idle: LogInfo("User %@ finished typing!", sender);

break;

}

});

}

### 负载自定义消息体

聊天消息支持扩展的消息格式数据传输。为此需要创建自定义媒体消息和更新自定义媒体消息如下：

KandyRecord \* kandyRecord = [[KandyRecord alloc] initWithURI:@" user1@kandy.io"];

// Create chat message. Custom users data can be added to any type of chat messages KandyChatMessage \*textMessage = [[KandyChatMessage alloc] initWithText:@Message text@ recipient:kandyRecord];

// Add custom data to message

[textMessage.mediaItem updateAddtitionalData:@{@"timestamp": @( [NDate timeIntervalSinceReferenceDate])}];

### 发送短信消息

发送短信消息到通常意义上的移动手机，使用如下代码［kandy service 支持发送SMS消息］。

* + - * (void)sendSMS: {
      * KandySMSMessage \*smsMessage = [[KandySMSMessage alloc] initWithText:@”The SMS text” recipient:@”18009871234” displayName:@”SMS display name”];
      * [[Kandy sharedInstance].services.chat sendSMS:smsMessage responseCallback:^(NSError \*error) {
      * if (error)
      * { //Failure }
      * else { //Success }
      * }];
      * }

注意：在短信前，请不要忘记添加国家代码在发送目的地手机号码。

### Pull事件

主动拉去Pending 聊天事件，使用如下代码：

-(void)pullEvents {

[[Kandy sharedInstance].services.chat pullEventsWithResponseCallback:^(NSError \*error) {

if (error) { // error }

}];

}

### 确认事件

在KandySDK 中可以通过发送确认事件来响应KandyChatServiceNotificationDelegate 聊天消息通知。在前面已经提及，现在再复述一下：

\* For Chat received:

-(void)onMessageReceived:(KandyMessage\*)kandyMessage { [kandyMessage markAsReceivedWithResponseCallback:^(NSError \*error) {

if (error) { //Failure

} else { //Success }

}];

}

\* For Chat delivered:

-(void)onMessageDelivered:(KandyDeliveryAck\*)ackData{ [[Kandy sharedInstance].services.chat markAsReceived:ackData withResponseCallback:^(NSError \*error) {

if (error) { //Failure

} else { //Success

}

}

]

];

### 下载媒体消息

下载媒体消息的代码如下：

-(void)downloadMediaForMessage:( id<KandyMessageProtocol>)kandyMessage {

[[Kandy sharedInstance].services.chat downloadMedia:kandyMessage progressCallback:^(KandyTransferProgress \*transferProgress) { //Download progress }

responseCallback:^(NSError \*error, NSString \*fileAbsolutePath) {

if (error) { //Failure

} else { //Success }

}];

}

### 下载媒体封面

下载媒体消息封面，代码如下：

-(void)downloadMediaThumbnail:( id<KandyMessageProtocol>)kandyMessage {

[[Kandy sharedInstance].services.chat downloadMediaThumbnail:kandyMessage thumbnailSize:EKandyThumbnailSize\_medium progressCallback:^(KandyTransferProgress \*transferProgress) { //Progress }

responseCallback:^(NSError \*error, NSString \*fileAbsolutePath) {

if (error) { //Failure

} else { //Success }

}];

}

### 取消媒体操作

取消上传／下载kandy 消息的操作，代码如下：

-(void)cancelOperation:( id<KandyMessageProtocol>)kandyMessage {

[[Kandy sharedInstance].services.chat cancelWithKandyMessage:kandyMessage responseCallback:^(NSError \*error) {

if (error) { //Failure

} else { //Success }

}];

}

### 设置相关

如下是聊天的相关设置：

autoDownload\_media\_connectionType：设置网络条件下app可以下载接受的媒体消息［仅3G，仅WiFi，总是，从不］。

autoDownload\_media\_maxSizeKB：下载媒体文件最大大小

autoDownload\_thumbnail：设置封面是否可自动下载

autoDownload\_thumbnailSize：自动下载封面文件的大小

downloadPathBuilder：复写的协议，设置下载文件路径和文件名称。

mediaMaxUploadSizeInKB：最大允许上传的文件大小。

设置kandy 聊天设置的代码如下：

[[Kandy sharedInstance].services.chat.settings

## 群组消息

本章节中，将介绍创建和管理群组，获取群组消息，发送群组消息和接受群组消息。

确保你已开启推送功能，在应用处于非激活状态时将收到新的消息推送。

### 监听群组状态

注册KandyGroupServiceNotificationDelegate，将收到群组状态改变通知，如：onGroupDestroyed, onGroupUpdated, onParticipantJoined等。

注册KandyGroupServiceNotificationDelegate 代理的代码如下：

[[Kandy sharedInstance].services.group registerServiceNotificationsDelegate:self];

解除注册如在［ViewDidDisappear］代码如下：

[[Kandy sharedInstance].services.group unregisterNotifications:self];

代理函数：

-(void)onGroupDestroyed:(KandyGroupDestroyed\*)groupDestroyedEvent{ //your code here }

-(void)onGroupUpdated:(KandyGroupUpdated\*)groupUpdatedEvent{ //your code here }

-(void)onParticipantJoined:(KandyGroupParticipantJoined\*) groupParticipantJoinedEvent{ //your code here }

-(void)onParticipantKicked:(KandyGroupParticipantKicked\*) groupParticipantKickedEvent{ //your code here }

-(void)onParticipantLeft:(KandyGroupParticipantLeft\*) groupParticipantLeftEvent{ //your code here }

### 创建群组

创建kandy群组并备注名称［可选］和头像［可选］

-(void)createGroup {

KandyGroupParams \* kandyGroupParams = [[KandyGroupParams alloc] init];

kandyGroupParams.groupName = @"groupName";

kandyGroupParams.groupAbsoluteImagePath = @"pathToGroupImage"; //will also initialize the group.image

[[Kandy sharedInstance].services.group createGroup:kandyGroupParams progressCallback:^(KandyTransferProgress \*transferProgress) { NSLog(@"Group Image upload progress : %ld", (long)transferProgress.transferProgressPercentage); }

responseCallback:^(NSError \*error, KandyGroup \*group) {

if (error) { //error

} else { //group created successfully

}};

}

### 添加／删除群组成员

添加／删除群组的参与者

-(void)addParticipant{

KandyRecord \* participant = [[KandyRecord alloc] initWithURI: @"participantURI”];

KandyRecord \* groupId = self.kandyGroup.groupId;

[[Kandy sharedInstance].services.group addGroupParticipants:@[participant] groupId:groupId responseCallback:^(NSError \*error, KandyGroup \*group) {

if (error) { //error

} else { //participants added

}

}];

}

### 群组状态更新

更新群组的名称和图片。如下函数可以删除群组名称和图片。

kRemoveGroupName 删除群组名称

kRemoveGroupImage 删除群组图片

-(void)createGroup{

KandyRecord \* groupId = self.kandyGroup.groupId;

KandyGroupParams \* params = [[KandyGroupParams alloc] init];

kandyGroupParams.groupName = @"groupName"; kandyGroupParams.groupAbsoluteImagePath = @"pathToGroupImage";

//will also initialize the group.image

[[Kandy sharedInstance].services.group updateGroupParams:params groupId:groupId progressCallback:^(KandyTransferProgress \*transferProgress) { NSLog(@"Update group image progress : %ld", (long)transferProgress.transferProgressPercentage); }

responseCallback:^(NSError \*error, KandyGroup \*group) { if (error) { //error

} else { //group created successfully }

}];

}

提示：

1. 更新群组名称和图片仅仅实在calling中
2. 删除群组图片也是可以在calling中

### 销毁群组

删除群组消息，代码如下：

-(void)destroyGroup{

[[Kandy sharedInstance].services.group destroyGroup:self.kandyGroup.groupId responseCallback:^(NSError \*error) {

if (error) { //error

} else { //group destryed

}

}];

}

### 我的群组

获取我的群组，代码如下：

-(void)getMyGroups{

[[Kandy sharedInstance].services.group getMyGroupsWithResponseCallback:^(NSError \*error, NSArray \*groups) { if (error) { //error

} else {

／/NSLog(@”My groups: %@”,groups);

}

}];

}

### 群组详情

获取群组详情，代码如下：

-(void)getGroupDetails{

[[Kandy sharedInstance].services.group getGroupDetails:self.groupId responseCallback:^(NSError \*error, KandyGroup \*group) {

if (error) { //error

} else {

//NSLog(@ “Group details: %@”,group);

}

}];

}

### 群组设置

如下是群组消息的相关设置：

downloadPathBuilder：复写的协议，设置下载文件路径和文件名称。

groupImageMaxUploadSizeInKB：最大允许上传的文件大小。

设置kandy 聊天设置的代码如下：

[[Kandy sharedInstance].services.group.settings

## 事件

事件服务可以推送当前的事件或则从服务器上获取事件历史。这个事件可以是聊天消息、群组消息、群组状态改变事件和传递推送消息。

### 监听服务器事件

接受历史事件监听函数［onConversationsDeleted, onHistoryEventsDeleted］。

通过实现KandyEventServiceNotificationDelegate函数可以接受通知消息。

收到事件消息后，我们应该确认事件消息，否则客户端将再次收到消息。

#### 注册监听事件

-(void)registerForHistoryEvents{

[[Kandy sharedInstance].services.events registerNotifications:self];

}

#### 解除监听事件

如下代码段实在ViewDidDisappear 中解除监听函数

* + - * (void) unRegisterForHistoryEvents {
      * [[Kandy sharedInstance].services.events unregisterNotifications:self];
      * }

#### 其他代理函数

-(void)onConversationsDeleted:(KandyConversationsDeleted \*)conversationsDeletedEvent {

// Handle your logic when some conversations has been deleted from history // Remove from list of pending events

[conversationsDeletedEvent markAsReceivedWithResponseCallback:nil];

}

-(void)onHistoryEventsDeleted:(KandyHistoryEventsDeleted \*)hisotryEventsDeletedEvent {

// Handle your logic when some events has been deleted from history

// Remove from list of pending events

[hisotryEventsDeletedEvent markAsReceivedWithResponseCallback:nil];

}

### 拉取 Pending 事件

如下是获取服务器事件的代码

从服务器获取的事件，将被自动分配到相关的服务代理对象上，支持聊天／群组／呼叫服务。

-(void)pullEvents{

[[Kandy sharedInstance].services.events pullPendingEventsWithResponseCallback:^(NSError \*error, NSArray \*kandyEvents) {

if(error) {

//Failure

} else {

//Success

}

}];

}

### 确认Pending 事件

在拉取Pending事件成功后应该发送这些事件的确认，否则在下次的拉去Pending事件还是会拉取掉。

标示事件已经接受并且不在以后的拉去Pending事件中获取了，调用：markAsReceivedWithResponseCallback： 在KandyEventProtocol：

-(void)onMessageReceived:(KandyMessage\*)kandyMessage {

[kandyMessage markAsReceivedWithResponseCallback:^(NSError \*error) {

if(error){ //Failure }

else

{ //Success }

}];

}

我们可以调用KandyEventService中的 markEventsAsReceived:responseCallback:

函数，而一次性确认多个Pending事件。

#### 对于聊天消息

-(void)onMessageReceived:(KandyMessage\*)kandyMessage {

[[Kandy sharedInstance].services.events markEventsAsReceived:@[kandyMessage] responseCallback:^(NSError \*error) {

if(error){

//Failure

} else {

//Success }

}];

}

对于投递结果消息

-(void)onMessageDelivered:(KandyDeliveryAck\*)ackData{

[[Kandy sharedInstance].services.events markEventsAsReceived: @[ackData] responseCallback:^(NSError \*error){

if(error) {

//Failure

} else {

//Success

}

}];

}

## 在线状态检测

在线状态检测可以获取联系人最近在线状态的时间。

### 最近在线状态

调用函数retrievePresenceForRecords:withResponseCallback可以获取最近在线状态。

<KandyPresenceProtocol> 中包装了在线状态检测相关函数，包括获取一组联系人的最近在线状态。

返回的callback是missingPresenceKandyRecords

结构的数组，标示当前不在线的用户及他们最近一次在线日期。

* + - * (void)getLastSeen{
      * KandyRecord\* kandyRecord = [[KandyRecord alloc] initWithURI:@"Contact@DomainName.Com" type:EKandyRecordType\_contact];
      * [[Kandy sharedInstance].services.presence getPresenceForRecords:[NSArray arrayWithObject:kandyRecord] responseCallback:^(NSError \*error, NSArray \*presenceObjects, NSArray \*missingPresenceKandyRecords) {
      * if (error) { // Failure
      * } else { // Success
      * id <KandyPresenceProtocol> presenceObject = [presenceObjects objectAtIndex:0];
      * NSString \*strDate = [NSDateFormatter localizedStringFromDate:presenceObject.lastSeen dateStyle:NSDateFormatterShortStyle timeStyle:NSDateFormatterShortStyle];
      * self.lblLastSeen.text = strDate;
      * }
      * }];
      * }

### 更新在线状态

在线状态支持用户设置用户在线状态，如：离开、工作中等。

更新在线状态调用如下：

[[Kandy sharedInstance].services.presence updateSelfPresenceState:isOnline presenceActivityType:presenceStatusType

responseCallback:^(NSError \*error) {

if (error) {

//update state failed

} else {

//update state succeeded

}

}];

### 开始监听在线状态

开始监听用户在线状态代码如下：

* + - * (void)startWatch {
      * KandyRecord\* kandyRecord = [[KandyRecord alloc] initWithURI:user\_id type:EKandyRecordType\_contact];
      * [[Kandy sharedInstance].services.presence startWatchPresenceForRecords:[NSArray arrayWithObject:kandyRecord] responseCallback:^(NSError \*error) {
      * UIAlertView \*alert;
      * if (error) { //start watch failed
      * } else { //start watchn succeeded
      * }
      * }];
      * }

### 停止监听在线状态

停止监听用户在线状态代码如下：

* + - * (void)stopWatch {
      * KandyRecord\* kandyRecord = [[KandyRecord alloc] initWithURI:user\_id type:EKandyRecordType\_contact];
      * [[Kandy sharedInstance].services.presence startWatchPresenceForRecords:[NSArray arrayWithObject:kandyRecord] responseCallback:^(NSError \*error) {
      * UIAlertView \*alert;
      * if (error) { //start watch failed
      * } else { //start watch succeeded
      * }
      * }];
      * }

## 通讯录管理

通讯录管理支持获取服务器域和本地通讯录列表，搜索通讯录列表和监听通讯改变事件。

通过实现KandyContactsServiceNotificationDelegate代理，可以接受通讯录改变的相关事件。

注册通知

-(void)registerForContactsChanges{

[[Kandy sharedInstance].services.contacts registerNotifications:self];

}

解除注册

-(void)registerForContactsChanges{

[[Kandy sharedInstance].services.contacts unregisterNotifications:self];

}

### 检索设备的通讯录

如下代码获取设备的通讯录列表以 id<KandyContactProtocol> 对象。如操作失败则返回error。

-(void)getDeviceContacts{

[[Kandy sharedInstance].services.contacts getDeviceContactsWithResponseCallback:^(NSError \*error, NSArray \*kandyContacts) {

if (error) { // Failure

} else { // Success }

}];

}

### 检索搜索设备通讯录

如下代码是搜索设备的通讯录列表以 id<KandyContactProtocol> 对象。如操作失败则返回error。

支持两个种搜索通讯录的方式：

1. 搜索联系人字段：phones／emails/all（按位操作）
2. 用户名称，在给予的搜索关键字段中。如果关键字为空，搜索将不起作用。

-(void)filterDeviceContacts{

[[Kandy sharedInstance].services.contacts getDeviceContactsWithFilter:EKandyDeviceContactFilter\_phone | EKandyDeviceContactFilter\_email textSearch:@"John" responseCallback:^(NSError \*error, NSArray \*kandyContacts) { if (error) { // Failure

} else { // Success

}

}];

}

### 检索域通讯录

如下代检索服务器域中通讯录列表以 id<KandyContactProtocol> 对象。如操作失败则返回error。

-(void)getDomainDirectoryContacts {

[[Kandy sharedInstance].services.contacts getDomainDirectoryContactsWithResponseCallback:^(NSError \*error, NSArray \*kandyContacts) {

if (error) { // Failure

} else { // Success

}

}];

}

### 查询域通讯录

如下代码是搜索服务器域的通讯录列表以 id<KandyContactProtocol> 对象。如操作失败则返回error。

Filtering是给予的搜索关键字，EKandyDomainContactFilter是标示是否开启过滤功能。

-(void)filterDomainDirectoryContacts {

[[Kandy sharedInstance].services.contacts getFilteredDomainDirectoryContactsWithTextSearch:@"John" filterType:EKandyDomainContactFilter\_firstAndLastName caseSensitive:NO responseCallback:^(NSError \*error, NSArray \*kandyContacts) {

if (error) { // Failure

} else { // Success

}

}];

}

### VCard名片

KandySDK提供便捷的方式来实现KandyContact和Vcard对象的转化。

#### KandyContact -> VCard

-(void)getVCardForKandyContact:(id<KandyContactProtocol>)kandyContact {

[[Kandy sharedInstance].services.contacts createVCardDataByContact:kandyContact completionBlock:^(NSError \*error, NSData \*vCardData) {

if (error) { // Failure

} else { // Success

}

}];

}

#### VCard ->KandyContact

-(id<KandyContactProtocol>)getKandyContactByVCard:( NSData\*)vCardData {

NSError\* error = nil;

[[Kandy sharedInstance].services.contacts createContactByVCard:vCardData error:&error];

if (error) { // Failure

} else { // Success

}

}

## 云存储

云存储服务包装了简单方法去上传／下载文件到Kandy 云存储空间中。

### 上传文件

上传文件代码

-(IBAction)uploadFile:(id)sender{

// Get file name and path for selected file

id<KandyFileItemProtocol> fileItem = [KandyMessageBuilder createFileItem:@”localPathFile” text:nil];

// Upload the file

[[Kandy sharedInstance].services.cloudStorage uploadMedia: fileItem progressCallback:^(KandyTransferProgress \*transferProgress) {

// Notify UI about progress change }

responseCallback:^(NSError \*error) {

if (error) { // Fail

} else {

// Success. Now you can get ContentUUID from updated fileItem to download your file later

}

}];

}

### 下载文件

从云存储中下载文件代码如下：

-(IBAction)downloadFile:(id)sender{

NSString \* downloadFilename = <Path and file name>;

id<KandyFileItemProtocol> transferingFileItem = [KandyMessageBuilder createFileItem:nil text:nil];

NSString \* contentUUID = <The file ContentUUID you wish to download>; [transferingFileItem updateContentUUID:contentUUID];

// Download the file

[[Kandy sharedInstance].services.cloudStorage downloadMedia:transferingFileItem fileName:[downloadFileName lastPathComponent] downloadPath:self.documentsDirectory progressCallback:^(KandyTransferProgress \*transferProgress) {

// Notify UI about progress change

} responseCallback:^(NSError \*error, NSString \*filePath) {

if (error) { // Fail

} else {

// Success. transferingFileItem now filled with all relevant information }

}];

}

### 下载封面

下载媒体封面的代码如下：（封面仅支持在图片和视频文件中）

-(IBAction)downloadFile:(id)sender{

NSString \* thumbnailFilename = <Path and file name>; id<KandyFileItemProtocol> fileItem = [KandyMessageBuilder createFileItem:thumbnailFileName text:nil];

NSString \* contentUUID = < The file ContentUUID you wish to download its thumbnail>;

[transferingFileItem updateContentUUID:contentUUID]; // Download the file thumbnail

[[Kandy sharedInstance].services.cloudStorage downloadThumbnail:fileItem thumbnailName:[ thumbnailFilename lastPathComponent] downloadPath:self.documentsDirectory thumbnailSize:EKandyThumbnailSize\_medium progressCallback:^(KandyTransferProgress \*transferProgress) {

// Notify UI about progress change }

responseCallback:^(NSError \*error, NSString \*filePath) {

if (error) { // Fail

} else { // Success

}

}];

}

### 取消操作

取消上传／下载过程代码如下：

-(IBAction)cancelTransfer:(id)sender{

[[Kandy sharedInstance].services.cloudStorage cancelMediaTransfer:self.transferingFileItem responseCallback:^(NSError \*error) {

if (error) { // Fail

} else { // Success }

}];

}

## 错误处理

SDK中错误的返回都包含两个字段：

\* Domain

\* Code

在每个场景中，domain可能是KandyGeneralServiceErrorDomain，或者操作的相关域，例如：KandyCallServiceErrorDomain

在每个domain中，有多种错误code，他们代表每种sdk识别的场景。

在错误处理中，应该先处理demain，然后匹配不同的错误code。

-(void)errorHandling {

[[Kandy sharedInstance].provisioning signoutWithResponseCallback:^(NSError \*error) {

BOOL isHandledError = YES;

if ([error.domain isEqualToString:KandyGeneralServiceErrorDomain]) { switch (error.code) {

case EKandyGeneralServiceError\_NetworkError: // Error handling code break;

case EKandyGeneralServiceError\_RequestTimeout: // Error handling code break;

default:

isHandledError = NO;

break; }

}

else if ([error.domain isEqualToString:KandySignupServiceErrorDomain]) {

switch (error.code) {

case EKandySignupErrors\_userValidationFailed: // Error handling code

case EKandySignupErrors\_validationCodeExpired: // Error handling code // . // . // . default:

isHandledError = NO;

break; }

}

else {

isHandledError = NO;

}

}];

}