# VOIP SDK 控件化框架集成指南

# iOS 版本

**Version 1.0**

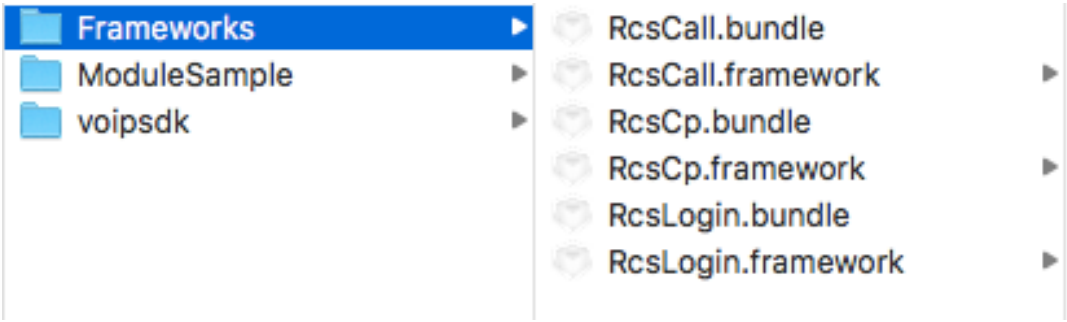
宁波菊风系统软件有限公司

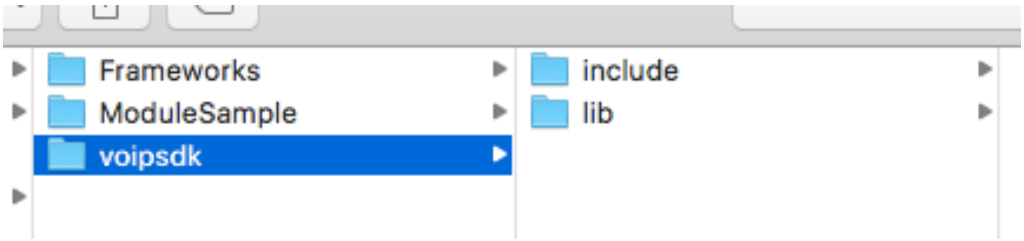
| 文档版本 | 更新人 | 更新时间 |
| --- | --- | --- |
| 1.0 | Ginger,Jeff | 2017-06-03 |
| 1.1 | Lith,Ginger | 2017-12-13 |

# 1 环境配置

## 1.1 新建路径

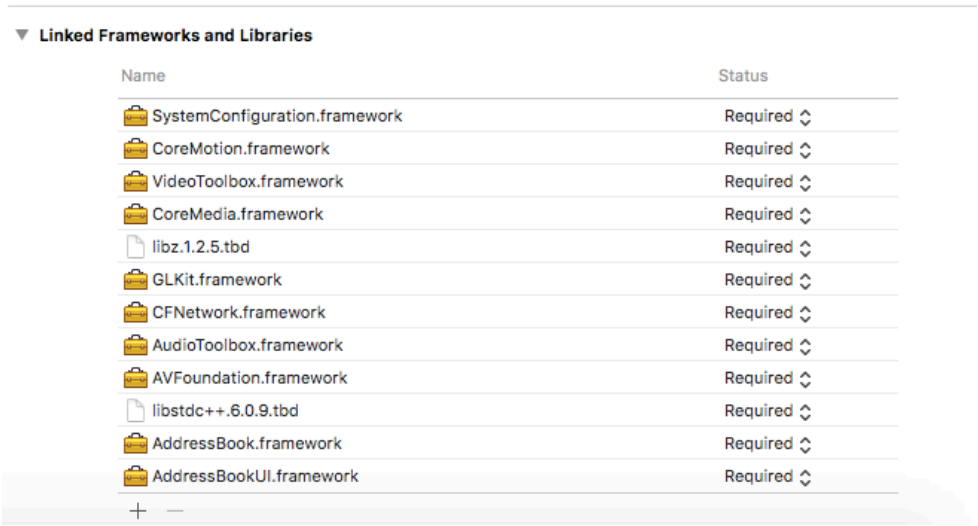
**在项目的根目录下创建 Frameworks 和 sdk。其中 Frameworks 中放入所需模块的.framework 和.bundle 文件（如 RcsLogin.framework、RcsLogin.bundle）。**




## 1.2 添加系统库

**General 中的 Linked Frameworks and Libraries 中添加以下的系统库**




## 1.3 添加 VOIP SDK 控件化框架

**随后将要集成的模块的.framework 和.bundle 通过 Add Other…的方式导入到项目中。**


## 1.4 Build Settings

1. Other Linker Flags 添加 -ObjC -lstdc++ -llemon -lmme_jrtc -lavatar -lresolv -lcrypto -larchive -lzmf -lzvs -lssl
2. Build Active Architecture Only 改为 NO
3. Enable Bitcode 改为 NO
4. Framework Search Paths 添加$(PROJECT_DIR)/../Frameworks
5. Header Search Paths 添加$(PROJECT_DIR)/../voipsdk/include 和$(PROJECT_DIR)/../voipsdk/include/zos
6. Library Search Paths 添加$(PROJECT_DIR)/../voipsdk/lib/ios
7. Preprocessor Macros 的 Debug 和 Release 均添加 ZPLATFORM=ZPLATFORM_IOS
（如已存在 PCH 文件并已引用，无需操作该步骤）新建 PCH 文件，命名为 PrefixHeader.pch，
8. Prefix Header 添加$(SRCROOT)/$(PROJECT_NAME)/PrefixHeader.pch

9.在 PrefixHeader 里#import “service/voip/mtc.h” #import “mme/zmf.h”，方便整个项目结合 SDK 的开发

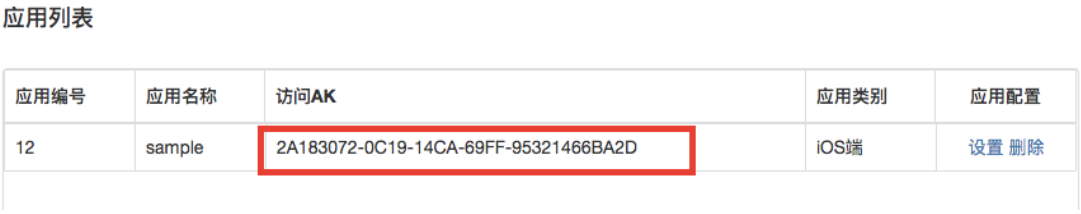
### 1.5 Lincense

将 lincense 文件直接加入到工程中，无路径要求

# 2 初始化

## 2.1 获取 appKey

访问 http://120.27.139.73:8087/developer，注册账号并登录，根据项目的类型及所需的功能勾选生成 appKey


红框内的即为获取到的 APPKey

## 2.2 Init

JusLoginDelegate 的 InitWithKey 方法里包含了 License 的检查和 SDK 的初始化、各路径的设置以及 APPKey 的设置，是其他模块集成开发的前提。
其他各模块 Jus***Delegate 的 Init 方法需在使用该模块前调用 Init，如没有则说明没有初始化的要求。

### 2.2.1 PCH 文件中需引入相关的头文件

```
#import "service/voip/mtc.h"
#import "mme/zmf.h"
#import <RcsLogin/RcsLogin.h>
#import <RcsCall/RcsCall.h>
```

### 2.2.2 初始化代码

-(BOOL)application:(UIApplication*)application didFinishLaunchingWithOptions:(NS Dictionary *)launchOptions 中进行初始化

```
[JusLoginDelegate SetJusInitListener:self];
    int result = [JusLoginDelegate InitWithKey:APP_KEY];

    switch (result) {
        case RESULT_LICENSE_EXPIRED:
            NSLog(@"%@",@"License文件有效期过期");
            break;
        case RESULT_LICENSE_NOT_EXIST:
            NSLog(@"%@",@"License文件不存在");
            break;
        case RESULT_MOD_REGISTER_ERROR:
            NSLog(@"%@",@"组件化注册失败");
            break;
        case RESULT_SDK_FLODER_CREATE_ERROR:
            NSLog(@"%@",@"SDK文件夹创建失败");
            break;
        case RESULT_LICENSE_GET_DEVICEID_FAILED:
            NSLog(@"%@",@"获取设备号失败");
            break;
        case RESULT_LICENSE_DOWNLOAD_START_FAILED:
            NSLog(@"%@",@"启动下载License失败");
            break;
        case RESULT_SUCCESS:
            [JusCallDelegate Init];
            [JusCallDelegate SetJusCallListener:self];
            break;
    }
```

## 2.3 添加 Listener 监听

Listener 为各模块的协议，设置代理即可收到各模块各流程的回调。在各模块 Init 之后一般

情况下都需设置代理。如 JusInitListener 包含 License 在线方案的流程监听。详细需求可参考源码注释来选择要实现的 Listener。

- 比如当前项目集成 RcsLogin 和 RcsCall 的 Framework，则可以让 AppDelegate 遵循 JusInitListener 和 JusCallListener 来监听初始化和通话流程的回调

```
@interface AppDelegate () <JusCallListener,JusInitListener>
{
```

# 3 控件化 Framework 一级接口使用指南

使用一级接口进行开发（适用无界面自定义需求的开发者）

## 3.1 RcsLogin

➢ 初始化 Login 注册模块：

```
[JusLoginDelegaet InitWithKey:APP_KEY];
```

➢ 注册状态监听

```
[[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(onRegisterStateChanged:) name:RcsRegStateChangedNotification object:nil];

- (void)onRegisterStateChanged:(NSNotification *)notification
{
    NSDictionary* useInfo = notification.userInfo;
    int state = [[useInfo objectForKey:RcsRegStatKey] intValue];
    int statCode = [[useInfo objectForKey:RcsStatCodeKey] intValue];
    NSString* statMessage = [useInfo objectForKey:RcsStatMessageKey];

    if (state == MTC_REG_STATE_REGED) {
        [self setEnableView:NO];
        [self.loginingIndicator stopAnimating];
        statMessage = @"登录成功";
    } else if (state == MTC_REG_STATE_IDLE){
        [self setEnableView:YES];

        switch (statCode) {
            case MTC_CLI_REG_ERR_DEACTED:
                [[[UIAlertView alloc] initWithTitle:statMessage
                                            message:nil delegate:self
```

```
                                        cancelButtonTitle:kStringCancel
                                        otherButtonTitles:kStringOk, nil] show];
                    break;
                case MTC_CLI_REG_ERR_REJECTED:

                    [[[UIAlertView alloc] initWithTitle:statMessage
                                                message:nil delegate:nil
                                        cancelButtonTitle:kStringOk
                                        otherButtonTitles:nil] show];
                    break;
            }

        } else if (state == MTC_REG_STATE_REGING){
            [self setEnableView:NO];
            statMessage = @"登录中...";
        } else if (state == MTC_REG_STATE_UNREGING){
            statMessage = @"注销中...";
        }
        self.navigationItem.title = statMessage;
}
```

**State**

```
/** @brief MTC state of REGISTER. */
#define MTC_REG_STATE_IDLE      0    /**< @brief Register idle state */
#define MTC_REG_STATE_REGING    1    /**< @brief Register registering state */
#define MTC_REG_STATE_REGED     2    /**< @brief Register registered state */
#define MTC_REG_STATE_UNREGING  3    /**< @brief Register unregistering state */
```

**dwstatCode**

```
/** @brief MTC status code of REGISTER. */
#define MTC_CLI_ERR_NO                  (MTC_EBASE_REG + 0) /**< @brief No error. */
#define MTC_CLI_ERR_LCL_FAILED          (MTC_EBASE_REG + 1) /**< @brief Local request error. */
#define MTC_CLI_REG_ERR_SEND_MSG        (MTC_EBASE_REG + 2) /**< @brief Send message error. */
#define MTC_CLI_REG_ERR_AUTH_FAILED     (MTC_EBASE_REG + 3) /**< @brief Register authentication failed, invalid user or password. */
#define MTC_CLI_REG_ERR_INVALID_USER    (MTC_EBASE_REG + 4) /**< @brief Register using invalid user. */
#define MTC_CLI_REG_ERR_TIMEOUT         (MTC_EBASE_REG + 5) /**< @brief Register timeout. */
#define MTC_CLI_REG_ERR_SERV_BUSY       (MTC_EBASE_REG + 6) /**< @brief Register server busy. */
#define MTC_CLI_REG_ERR_SERV_NOT_REACH  (MTC_EBASE_REG + 7) /**< @brief Register server not reached. */
#define MTC_CLI_REG_ERR_SRV_FORBIDDEN   (MTC_EBASE_REG + 8) /**< @brief Register forbidden. */
#define MTC_CLI_REG_ERR_SRV_UNAVAIL     (MTC_EBASE_REG + 9) /**< @brief Register unavailable. */
#define MTC_CLI_REG_ERR_DNS_QRY         (MTC_EBASE_REG + 10) /**< @brief Register dns query error. */
#define MTC_CLI_REG_ERR_NETWORK         (MTC_EBASE_REG + 11) /**< @brief Register network error. */
#define MTC_CLI_REG_ERR_DEACTED         (MTC_EBASE_REG + 12) /**< @brief Register deactived. */
#define MTC_CLI_REG_ERR_PROBATION       (MTC_EBASE_REG + 13) /**< @brief Register probation */
#define MTC_CLI_REG_ERR_INTERNAL        (MTC_EBASE_REG + 14) /**< @brief Register internal error. */
#define MTC_CLI_REG_ERR_NO_RESOURCE     (MTC_EBASE_REG + 15) /**< @brief Register no resource. */
#define MTC_CLI_REG_ERR_REJECTED        (MTC_EBASE_REG + 16) /**< @brief Register be rejected. */
#define MTC_CLI_REG_ERR_SIP_SESS        (MTC_EBASE_REG + 17) /**< @brief Register sip session error. */
#define MTC_CLI_REG_ERR_UNREG           (MTC_EBASE_REG + 18) /**< @brief Register stop or unregsiter error. */
#define MTC_CLI_REG_ERR_INVALID_ADDR    (MTC_EBASE_REG + 19) /**< @brief Register using invalid ip addr. */
#define MTC_CLI_REG_ERR_WAIT_PWD        (MTC_EBASE_REG + 20) /**< @brief Register wait prompt password timeout. */
#define MTC_CLI_REG_ERR_NOT_FOUND       (MTC_EBASE_REG + 21) /**< @brief Register not found user. */
#define MTC_CLI_REG_ERR_AUTH_REJECT     (MTC_EBASE_REG + 22) /**< @brief Register authentication rejected, rorbidden to user register in hss. */
#define MTC_CLI_REG_ERR_ID_NOT_MATCH    (MTC_EBASE_REG + 23) /**< @brief Register identities don't match, the pvi of user not inconformity. */
#define MTC_CLI_REG_ERR_USER_NOT_EXIST  (MTC_EBASE_REG + 24) /**< @brief Register user not exist. */
#define MTC_CLI_REG_ERR_OTHER           (MTC_EBASE_REG + 200) /**< @brief Other register error. */
```

- 发起注册

[JusLoginDelegate Login:(NSString *)user password:(NSString *)password authName:(NSString*)authName serverIP:(NSString *)serverIP serverRealm:(NSString*)serverRealm regTpt:(EN_MTC_TPT_

TYPE)regTpt regPort:(int)regPort regSrvType:(EN_MTC_REG_SRV_TYPE)regSrvType];
user：账号
password：密码
authName：鉴权名
serverIP：服务器地址（可 IP 或 Url 地址）
serverRealm：服务器域名
regTpt：传输类型（UDP/TCP/TLS）
regPort：注册端口
regSrvType：注册类型（普通 VoIP 用户：EN_MTC_REG_SRV_VOIP，标准 RCS 用户：EN_MTC_REG_SRV_JOYN_HF，CMCC RCS 用户，EN_MTC_REG_SRV_CMCC_RCS）

## 3.2 RcsCall

- 初始化 Call 通话模块

[JusCallDelegate Init];

- 发起一对一呼叫

[JusCallDelegate Call : (NSString *)number isVideo:(BOOL)isVideo];
number：对方号码
isVideo：是否为视频呼叫

- 发起多方通话

JusCallDelegate MultiCallWithNumbers: (NSArray*)peerNumbers;
peerNumbers： 号码数组

JusCallDelegate MultiCallWithMembers: (NSArray*)rcsCallMembers;
rcsCallMemebers： RcsCallMemeber 对象数组（带名字、头像等信息）

如有额外的需求，如添加通话记录等，则需要+ (void) SetJusCallListener:(id<JusCallListener>) listener;来设置代理监听整个通话过程，详见源码注释，同时建议将 Listener 设置为 AppDelegate，这样可以保证在程序启动时就完成所有的配置设置。

# 4 控件化 Framework 二级接口使用指南

## 4.1 RcsLogin

该模块功能包含：

- SDK 初始化的方法
- License 的检测及线上方案
- 复用的公用类

- 简单的账号相关页面

## 发起注册

**JusLoginDelegate 包含了可配置多种参数的注册方式，开发者可根据自己所需的配置选择注册方法。**

```
+ (BOOL) AutoLogin;//自动登录最后一个已登录状态的账号
+ (BOOL) Login;
+ (BOOL) Login:(NSString *)user;
+ (BOOL) Login:(NSString *)user password:(NSString *)password serverIP:(NSString *)serverIP;
+ (BOOL) Login:(NSString *)user password:(NSString *)password serverIP:(NSString *)serverIP regTpt:(EN_MTC_TPT_TYPE)regTpt regPort:(int)regPort;
+ (BOOL) Login:(NSString *)user password:(NSString *)password serverIP:(NSString *)serverIP regTpt:(EN_MTC_TPT_TYPE)regTpt regPort:(int)regPort regSrvType:(EN_MTC_REG_SRV_TYPE)regSrvType;

+ (BOOL) Login:(NSString *)user password:(NSString *)password serverIP:(NSString *)serverIP serverRealm:(NSString*)serverRealm regTpt:(EN_MTC_TPT_TYPE)regTpt regPort:(int)regPort regSrvType:(EN_MTC_REG_SRV_TYPE)regSrvType;
+ (BOOL) Login:(NSString *)user password:(NSString *)password authName:(NSString*)authName serverIP:(NSString *)serverIP serverRealm:(NSString*)serverRealm regTpt:(EN_MTC_TPT_TYPE)regTpt regPort:(int)regPort regSrvType:(EN_MTC_REG_SRV_TYPE)regSrvType;
```

## 监听注册状态

**开发者可通过监听 RcsRegStateChangedNotification 的广播来知晓注册状态的变更。**

```
[[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(onRegisterStateChanged:) name:RcsRegStateChangedNotification object:nil];

- (void)onRegisterStateChanged:(NSNotification *)notification
{
    NSDictionary* useInfo = notification.userInfo;
    int state = [[useInfo objectForKey:RcsRegStatKey] intValue];
    int statCode = [[useInfo objectForKey:RcsStatCodeKey] intValue];
    NSString* statMessage = [useInfo objectForKey:RcsStatMessageKey];

    if (state == MTC_REG_STATE_REGED) {
        [self setEnableView:NO];
        [self.logininglndicator stopAnimating];
        statMessage = @"登录成功";
    } else if (state == MTC_REG_STATE_IDLE){
        [self setEnableView:YES];
```

```
            switch (statCode) {
                case MTC_CLI_REG_ERR_DEACTED:
                    [[[UIAlertView alloc] initWithTitle:statMessage
                                                message:nil delegate:self
                                      cancelButtonTitle:kStringCancel
                                      otherButtonTitles:kStringOk, nil] show];
                    break;
                case MTC_CLI_REG_ERR_REJECTED:

                    [[[UIAlertView alloc] initWithTitle:statMessage
                                                message:nil delegate:nil
                                      cancelButtonTitle:kStringOk
                                      otherButtonTitles:nil] show];
                    break;
            }

        } else if (state == MTC_REG_STATE_REGING){
            [self setEnableView:NO];
            statMessage = @"登录中...";
        } else if (state == MTC_REG_STATE_UNREGING){
            statMessage = @"注销中...";
        }
        self.navigationItem.title = statMessage;
}
```

**State**

```
/** @brief MTC state of REGISTER. */
#define MTC_REG_STATE_IDLE      0    /**< @brief Register idle state */
#define MTC_REG_STATE_REGING    1    /**< @brief Register registering state */
#define MTC_REG_STATE_REGED     2    /**< @brief Register registered state */
#define MTC_REG_STATE_UNREGING  3    /**< @brief Register unregistering state */
```

**dwstatCode**

```
/** @brief MTC status code of REGISTER. */
#define MTC_CLI_ERR_NO                  (MTC_EBASE_REG + 0) /**< @brief No error. */
#define MTC_CLI_ERR_LCL_FAILED          (MTC_EBASE_REG + 1) /**< @brief Local request error. */
#define MTC_CLI_REG_ERR_SEND_MSG        (MTC_EBASE_REG + 2) /**< @brief Send message error. */
#define MTC_CLI_REG_ERR_AUTH_FAILED     (MTC_EBASE_REG + 3) /**< @brief Register authentication failed, invalid user or password. */
#define MTC_CLI_REG_ERR_INVALID_USER    (MTC_EBASE_REG + 4) /**< @brief Register using invalid user. */
#define MTC_CLI_REG_ERR_TIMEOUT         (MTC_EBASE_REG + 5) /**< @brief Register timeout. */
#define MTC_CLI_REG_ERR_SERV_BUSY       (MTC_EBASE_REG + 6) /**< @brief Register server busy. */
#define MTC_CLI_REG_ERR_SERV_NOT_REACH  (MTC_EBASE_REG + 7) /**< @brief Register server not reached. */
#define MTC_CLI_REG_ERR_SRV_FORBIDDEN   (MTC_EBASE_REG + 8) /**< @brief Register forbidden. */
#define MTC_CLI_REG_ERR_SRV_UNAVAIL     (MTC_EBASE_REG + 9) /**< @brief Register unavailable. */
#define MTC_CLI_REG_ERR_DNS_QRY         (MTC_EBASE_REG + 10) /**< @brief Register dns query error. */
#define MTC_CLI_REG_ERR_NETWORK         (MTC_EBASE_REG + 11) /**< @brief Register network error. */
#define MTC_CLI_REG_ERR_DEACTED         (MTC_EBASE_REG + 12) /**< @brief Register deactived. */
#define MTC_CLI_REG_ERR_PROBATION       (MTC_EBASE_REG + 13) /**< @brief Register probation */
#define MTC_CLI_REG_ERR_INTERNAL        (MTC_EBASE_REG + 14) /**< @brief Register internal error. */
#define MTC_CLI_REG_ERR_NO_RESOURCE     (MTC_EBASE_REG + 15) /**< @brief Register no resource. */
#define MTC_CLI_REG_ERR_REJECTED        (MTC_EBASE_REG + 16) /**< @brief Register be rejected. */
#define MTC_CLI_REG_ERR_SIP_SESS        (MTC_EBASE_REG + 17) /**< @brief Register sip session error. */
#define MTC_CLI_REG_ERR_UNREG           (MTC_EBASE_REG + 18) /**< @brief Register stop or unregsiter error. */
#define MTC_CLI_REG_ERR_INVALID_ADDR    (MTC_EBASE_REG + 19) /**< @brief Register using invalid ip addr. */
#define MTC_CLI_REG_ERR_WAIT_PWD        (MTC_EBASE_REG + 20) /**< @brief Register wait prompt password timeout. */
#define MTC_CLI_REG_ERR_NOT_FOUND       (MTC_EBASE_REG + 21) /**< @brief Register not found user. */
#define MTC_CLI_REG_ERR_AUTH_REJECT     (MTC_EBASE_REG + 22) /**< @brief Register authentication rejected, rorbidden to user register in hss. */
#define MTC_CLI_REG_ERR_ID_NOT_MATCH    (MTC_EBASE_REG + 23) /**< @brief Register identities don't match, the pvi of user not inconformity. */
#define MTC_CLI_REG_ERR_USER_NOT_EXIST  (MTC_EBASE_REG + 24) /**< @brief Register user not exist. */
#define MTC_CLI_REG_ERR_OTHER           (MTC_EBASE_REG + 200) /**< @brief Other register error. */
```

## 4.2 RcsCall (通话模块)

**自定义界面开发除通话界面需重新编写，其他流程都与使用一级接口开发一样。**

该模块包含功能：

- 一对一音视频通话
- 多方音频通话
- 呼叫等待
- 呼叫保持
- 通话中音视频互转
- 静音
- 一对一转多方通话
- 转接
- 多方通话邀请踢出成员
- 发送 DTMF
- 录音

### 4.2.1 一对一音频通话

#### 4.2.1.1 主叫方

调用 JusCallDelegate 的+ (void) Call : (NSString *)number isVideo:(BOOL)isVideo 方法，该方法会根据开发者是否设置了自定义的通话界面来判断由谁发起呼叫，默认为自带的 viewController。通过- (id<RcsCallDelegate>)jusCallViewController;来设置自定义的 CallViewController。

确定要发起的 CallViewController 之后将根据 CallViewController 中的- (void)callSessionCall:(NSString *)phone isVideo:(BOOL)isVideo 方法发起呼叫业务，所以要在方法中需创建业务对象 RcsCallSession 来完成业务发起，同时将 CallViewController present 出来。

```
- (id<RcsCallDelegate>)jusCallViewController
{
    CallViewController *callView = [[CallViewController alloc]init];
    return callView;
}
```

```
RcsCallSession *callSession = [[RcsCallManager sharedInstance] createCallSession];
[callSession call:phone callType:isVideo?EN_RCS_CALL_ONE_VIDEO:EN_RCS_CALL_ONE_VOICE];
```

呼叫发起后将进入通话流程会有回调上报，此处的回调分两类：

- RcsCallDelegate

所有业务的回调上报，用于更新 U。

- JusCallListener

通话进程的上报，用于通话记录的写入及更新。

- **作为主叫时 RcsCallDelegate 的回调流程**

```
- (void)callSessionProgressing:(RcsCallSession*)session
{
//响铃中等待对方接听，UI 提示用户正在振铃，根据 session 判断是否为视频通话来决定是否要显示视频画面
}

- (void)callSessionStarted:(RcsCallSession*)session
{
//对方接听了通话，这时 UI 应将所有可操作的业务按钮显示，并提示用户通话开始，计时器开始计时，如果是视频通话，则要将对方的视频画面进行显示
}

- (void)callSessionStartFailed:(RcsCallSession*)session  statCode:(ZUINT)statCode
{
//通话失败的回调，开发者可根据 stateCode 提醒用户原因
}

- (void)callSessionTerminated:(RcsCallSession*)session  statCode:(ZUINT)statCode
{
//通话结束，这时应将 RcsCallSession 释放，并将 CallViewController  dismiss
}
```

- **作为主叫时 JusCallListener 的回调流程**

```
- (void)jusCallOutgoing:(RcsCallSession*)callSession
{
//呼出，此时应插入一条通话记录
}

- (void)jusCallTalking:(RcsCallSession*)callSession
{
//通话建立，此时应更新通话记录为建立成功的通话
}

- (void)jusCallTermed:(RcsCallSession*)callSession
{
//通话结束，根据是否为建立成功来判断这次通话的性质
}
```

### 4.2.1.2 被叫方

收到来电后将会根据- (id<RcsCallDelegate>)jusCallViewController;判断是否设置了自定义的 CallViewController，如果没有，则使用默认的 CallViewController。

有接听，拒接两个选项。
拒接：[callSession reject: EN_MTC_CALL_TERM_REASON_TYPE];
接听：[callSession accept: EN_RCS_CALL_TYPE];

**1.作为被叫时 RcsCallDelegate 的回调流程**

```
- (void)callSessionIncoming:(RcsCallSession*)session ringFileName:(NSString*)ringFileName
{
    //收到来电，将 CallViewController present，并显示此次通话的相关信息，通过 ringfilename 播放铃声
    //通过判断 session 是否为视频通话来选择界面做何处理
}

- (void)callSessionStarted:(RcsCallSession*)session
{
    //接听后通话建立成功
    //这时 UI 应将所有可操作的业务按钮显示，并提示用户通话开始，计时器开始计时，
}

- (void)callSessionStartFailed:(RcsCallSession*)session statCode:(ZUINT)statCode
{
    //通话建立失败，开发者可根据 stateCode 提醒用户原因
}

- (void)callSessionTerminated:(RcsCallSession*)session statCode:(ZUINT)statCode
{
    //拒接或者挂断之后收到该回调
    //通话结束，这时应将 RcsCallSession 释放，并将 CallViewController dismiss
}
```

**2.作为被叫时 JusCallListener 的回调流程**

```
(BOOL)jusCallIncomingReceived:(RcsCallSession*)callSession
{
    //收到来电，此时可根据是否为黑名单号码来决定是否直接拒接
}

- (void)jusCallIncomingDone:(RcsCallSession*)callSession
{
    //进入正常的来电流程，此时可以插入来电信息，状态为未接
```

```
}

- (void)jusCallTalking:(RcsCallSession*)callSession
{
//开始通话，此时应更新来电信息为已接
}

- (void)jusCallTermed:(RcsCallSession*)callSession
{
//通话结束，根据之前是否接听来判断是否要显示为未接来电
}
```

### 4.2.2 一对一视频通话

一对一视频通话的流程与音频通话相同，不赘述。在此介绍下音频相关工具的使用。

#### 4.2.2.1 创建可渲染视图

显示预览或者远端的图像都要创建一个可渲染的视图，通过[[RcsCallManager shareInstance] creatZmfView:CGRect];即可创建传入参数大小的视图，然后根据需要将其放置显示。

#### 4.2.2.2 设置摄像头类型

[callSession.videoCallProvider setCameraType: RCS_CAMERA_TYPE];
调用该接口开启摄像头，RCS_CAMERA_FRONT 为前置，RCS_CAMERA_BACK 为后置。

#### 4.2.2.3 渲染预览视图

[callSession.videoCallProvider setPreviewSurfaceView:UIView *];
创建的可渲染视图设置为预览图，此时就会将预览画面显示于你所传入的视图中。

#### 4.2.2.4 渲染对端视图

[callSession.videoCallProvider setDisplaySurfaceView:UIView *];
创建的可渲染视图设置为远端图，此时就会将预览画面显示于你所传入的视图中。

#### 4.2.2.5 翻转摄像头

[callSession.videoCallProvider switchCamera:UIView *];

#### 4.2.2.6 移除预览图

[callSession.videoCallProvider removePreviewSurfaceView:UIView *];

#### 4.2.2.7 移除远端图

[callSession.videoCallProvider removeDisplaySurfaceView:UIView *];

#### 4.2.2.8 动画将预览图缩小

[callSession.videoCallProvider shrinkPreviewSurfaceView:UIView* parentView:UIView *]
用于得到对端画面后将预览图缩小

### 4.2.3 一对一音视频呼叫切换

#### 4.2.3.1 主叫方

- **音频切为视频**

[callSession update:EN_RCS_CALL_ONE_VIDEO];
同时应将自己的视频预览图开启。

- **视频切为音频**

[callSession update:EN_RCS_CALL_ONE_VOICE];
直接更新为音频通话界面

- **音视频切换回调**

```
-(void)callSessionUpdated:(RcsCallSession *)session{
    //若对方对你的音频转视频邀请做出响应，则会收到该回调
    //可以根据此时[session isVideo]来更新 UI 为视频通话界面或是音频通话界面
}
```

#### 4.2.3.2 被叫方收到音频切换视频的请求

只在音频切换到视频的时候询问，视频切换音频无请求
收到- (void)callSessionUpdateReceived:(RcsCallSession *)session 的回调
用[callSession responseUpdate:EN_RCS_CALL_ONE_VOICE]表示拒接切换
用[callSession responseUpdate: EN_RCS_CALL_ONE_VIDEO]表示接受切换
之后会收到-(void)callSessionUpdated:(RcsCallSession *)session 的回调，可以根据此时 session

的类型来更新 UI 为视频通话界面或是音频通话界面

### 4.2.4 呼叫保持

## 4.2.4.1 发起保持

```
[callSession  hold];
```

- **相关回调**

```
-(void)callSessionHoldOk:(RcsCallSession  *)session{
    //保持成功
    //此时可更新保持通话按钮的状态和停止定时器，或者提醒用户当前为保持状态。
}

-(void)callSessionHoldFailed:(RcsCallSession  *)session  statCode:(ZUINT)statCode  {
    //保持失败,提醒用户
}
```

## 4.2.4.2 被保持

```
-(void)callSessionHoldReceived:(RcsCallSession  *)session{
    //被保持方收到该回调
    //此时可更新保持通话按钮的状态和停止定时器，或者提醒用户当前为保持状态。
}
```

## 4.2.4.3 解除保持

```
[callSession  unhold];
```

- **相关回调**

```
-(void)callSessionUnHoldOk:(RcsCallSession  *)session{
    //解除保持成功
    //此时可更新保持通话按钮的状态和打开定时器，或者提醒用户当前为保持状态。
}

-(void)callSessionUnHoldFailed:(RcsCallSession  *)session  statCode:(ZUINT)statCode{
    //解除保持失败，提醒用户
}
```

#### 4.2.4.4 解除被保持

```
-(void)callSessionUnHoldReceived:(RcsCallSession *)session{
    //解除被保持成功
    //此时可更新保持通话按钮的状态和打开定时器，或者提醒用户当前为保持状态。
}
```

### 4.2.5 呼叫等待

在已有通话的情况下收到新来电会收到

```
-(void)callSessionWaiting:(RcsCallSession *)anotherSession ringFileName:(NSString *)ringFileName{
    //根据 ringfilename 播放铃声，根据 anotherSession 获取新一路通话的详细信息，提示用户是否要接受。
    //如果接受，就讲前一路通话保持[curSession hold];
    //接受：[anotherSession accept:EN_RCS_CALL_TYPE];
    //拒绝：[anotherSession reject:EN_MTC_CALL_TERM_REASON_TYPE];
    //后续回调流程与普通来电相同。

}
```

### 4.2.6 录音

- **开始录音**

```
[callSession recordStart];
```

- **触发回调**

```
- (void)callSessionRecordStartOK:(RcsCallSession*)session
{
    //开始录音成功
}
- (void)callSessionRecordStartFailed:(RcsCallSession*)session
{
    //开始录音失败
}
```

- **停止录音**

```
[callSession recordStop];
```

- **触发回调**

```
- (void)callSessionRecordStopOK:(RcsCallSession *)session
{
    //停止成功
```

```
}

- (void)callSessionRecordStopFailed:(RcsCallSession *)session
{
    //停止失败
}
```

### 4.2.7 呼叫后转

- **发起呼叫转移**

```
[callSession transf:peerNumber];
```

- **触发回调**

```
-(void)callSessionUpdated:(RcsCallSession *)session
{
    //转接被接受
}

- (void)callSessionStartFailed:(RcsCallSession*)session statCode:(ZUINT)statCode
{
    //转接失败
}
```

### 4.2.8 DTMF

```
[callSession sendDtmf:str];
```

支持 0~9 以及# *的字符 dtmf 发送

### 4.2.9 多方通话

#### 4.2.9.1 主叫方

调用 JusCallDelegate 的+ (void) MultiCallWithNumbers: (NSArray*)peerNumbers;或+ (void) MultiCallWithMembers: (NSArray*)rcsCallMembers;方法，该方法会根据开发者是否设置了自定义的通话界面来判断由谁发起呼叫，默认为自带的 viewController。通过- (id<RcsCallDelegate>)jusCallViewController;来设置自定义的 CallViewController。

确定要发起的 CallViewController 之后将根据 CallViewController 中的- (void)callSessionMultiCallWithMembers:(NSArray*)rcsCallMembers 方法发起呼叫业务，所以要在方法中创建业务对象 RcsCallSession 来完成业务发起，同时将 CallViewController present 出来。

```
- (id<RcsCallDelegate>)jusCallViewController
{
    CallViewController *callView = [[CallViewController alloc]init];
    return callView;
}
```

```
RcsCallSession *session = [[RcsCallManager sharedInstance] createCallSession];
[session multiCallWithMembers:rcsCallMembers];
```

- **作为主叫时 RcsCallDelegate 的回调流程**

- (void)callSessionProgressing:(RcsCallSession*)session
{
//响铃中等待对方接听，UI 提示用户正在振铃，根据 session 判断出是否为多方通话来选择 UI
}

- (void)callSessionStarted:(RcsCallSession*)session
{
//多方通话建立，这时 UI 应将所有可操作的业务按钮显示，并提示用户通话开始，计时器开始计时
}

- (void)callSessionStartFailed:(RcsCallSession*)session statCode:(ZUINT)statCode
{
//通话失败的回调，开发者可根据 stateCode 提醒用户原因
}

- (void)callSessionTerminated:(RcsCallSession*)session statCode:(ZUINT)statCode
{
//通话结束，这时应将 RcsCallSession 释放，并将 CallViewController dismiss
}

- **作为主叫时 JusCallListener 的回调流程**

- (void)jusConfOutgoing:(RcsCallSession*)callSession
{
//呼出多方通话，插入多方通话的通话记录
}

- (void)jusConfConned:(RcsCallSession*)callSession
{
//多方通话建立成功，更新通话状态
}

- (void)jusConfDisced:(RcsCallSession*)callSession

```
{
//多方通话结束，更新通话状态
}
```

### 4.2.9.2 被叫方

- **作为被叫时 RcsCallDelegate 的回调流程**

```
- (void)callSessionIncoming:(RcsCallSession*)session ringFileName:(NSString*)ringFileName
{
//收到来电，根据 session 判断为多方通话，播放铃声和更新 UI，将 CallViewController pres
ent
}

- (void)callSessionStarted:(RcsCallSession*)session
{
//多方通话开始，更新 UI
}

- (void)callSessionStartFailed:(RcsCallSession*)session statCode:(ZUINT)statCode
{
//多方通话开始失败，根据 statCode 提示用户
}

- (void)callSessionTerminated:(RcsCallSession*)session statCode:(ZUINT)statCode
{
//多方通话结束，释放 RcsCallSession，dismiss CallViewController
}
```

- **作为被叫时 JusCallListener 的回调流程**

```
- (BOOL)jusConfIncomingReceived:(RcsCallSession*)callSession
{
//收到多方通话来电，可根据是否是黑名单来决定是否直接拒接
}

- (void)jusConfIncomingDone:(RcsCallSession*)callSession
{
//若没有拒接，则插入多方通话来电的通话记录
}

- (void)jusConfConned:(RcsCallSession*)callSession
{
//多方通话建立成功，更新通话状态
```

```
}

- (void)jusConfDisced:(RcsCallSession*)callSession
{
//多方通话结束，更新通话状态，若未接听就挂断则为未接来电
}
```

## 4.2.9.3 多方通话成员管理

#### 4.2.9.3.1 邀请新成员

```
[callSession inviteParticipants:array];
```

- **RcsCallDelegate 的相关回调**

```
-(void)callSessionInviteParticipantsRequestDelivered:(RcsCallSession*)session
{
    //邀请发送成功
}

-(void)callSessionInviteParticipantsRequestFailed:(RcsCallSession*)session statCode:(ZUINT)statCode
{
    //邀请发送失败
}
```

- **JusCallDelegate 的相关回调**

```
- (void)jusConfIvtAcpt:(RcsCallSession*)callSession member:(RcsCallMember*)member
{
    //邀请成功
}
```

#### 4.2.9.3.2 踢出成员

```
[callSession removeParticipants:array]；
```

- **RcsCallDelegate 的相关回调**

```
- (void)callSessionRemoveParticipantsRequestDelivered:(RcsCallSession*)session
{
    //提出成员请求发送成功
}
-(void)callSessionRemoveParticipantsRequestFailed:(RcsCallSession*)session statCode:(ZUINT)statCode
```

```
{
    //提出成员请求发送失败
}
```

- **JusCallDelegate 的相关回调**

```
- (void)jusConfKickAcpt:(RcsCallSession*)callSession  member:(RcsCallMember*)member
{
    //提出成员成功
}
```

#### 4.2.9.3.3 成员信息变更

- **RcsCallDelegate 的相关回调**

```
-(void)callSessionConferenceStateUpdated:(RcsCallSession*)session{
    //可从 session 中获得 RcsCallMembers 对象进行 UI 的更新
}
```

- **JusCallDelegate 的回调流程**

```
- (void)jusConfUpdt:(RcsCallSession*)callSession  member:(RcsCallMember*)member
{
    //某个成员状态更新
}
```

#### 4.2.9.3.4 参与者

多方通话参与者一般不具备控制通话成员的权利，是否可以收到成员更新的通知也由服务器决定。若平台支持参与者成员更新通知，则参与者可在群成员状态改变时收到- (void)callSessionConferenceStateUpdated:(RcsCallSession*)session;回调。开发者可以在此回调中进行参与者通话界面的更新。

### 4.2.10 一对一通话转多方通话

## 4.2.10.1 两路通话合并为一路通话

当前已有一路通话时可以通过挂起当前通话发起新一路的通话，或是挂起当前通话接听另一个通话来实现两路通话。此时可以通过调用任一通话的[callSession merge]，来将两路通话合并为多方通话。

- **RcsCallDelegate 的主席回调流程**

```
-(void)callSessionMergeStarted:(RcsCallSession*)session  mergedSession:(RcsCallSession*)merge
dSession
{
    //合并开始
}

- (void)callSessionMergeComplete:(RcsCallSession*)session
{
    //合并成功
}

- (void)callSessionMergeFailed:(RcsCallSession*)session  statCode:(ZUINT)statCode
{
    //合并失败
}
```

- **RcsCallDelegate 的被叫回调**

```
- (void)callSessionConferenceExtendReceived:(RcsCallSession*)session
{
    //作为被叫被扩展为多方通话
}
```

- **JusCallDelegate 的主席方回调**

```
- (void)jusConfOutgoing:(RcsCallSession*)callSession
{
    //多方通话开始发起
}
- (void)jusConfConned:(RcsCallSession*)callSession
{
    //多方通话创建成功
}
- (void)jusConfDisced:(RcsCallSession*)callSession
{
    //多方通话结束
}
```

- **JusCallDelegate 的成员方回调**

```
- (void)jusCallToConf:(RcsCallSession*)callSession
{
    //一对一通话转为多方通话
}
```

### 4.2.10.2 一路通话扩展为多方通话

当前已有一路通话，可以邀请多人直接扩展为多方通话。
[callSession extendToConference:array];

- **RcsCallDelegate 的主席回调流程**

```
- (void)callSessionConferenceExtendStarted:(RcsCallSession*)session
{
    //扩展开始
}

- (void)callSessionConferenceExtendComplete:(RcsCallSession*)session
{
    //扩展成功
}

- (void)callSessionConferenceExtendFailed:(RcsCallSession*)session statCode:(ZUINT)statCode
{
    //扩展失败
}
```

- **RcsCallDelegate 的被叫回调**

```
- (void)callSessionConferenceExtendReceived:(RcsCallSession*)session
{
    //作为被叫被扩展为多方通话
}
```

- **JusCallDelegate 的主席方回调**

同两路通话合并为多方通话

- **JusCallDelegate 的成员方回调**

同两路通话合并为多方通话

# 5 DM 接口使用指南

## 5.1 进行必要的参数设置

[[RcsCpManager sharedInstance] setupAutoConfigWithUsername:user dmServerAddress:kServerAddress waitPromptTime:kCountDownTime];

参数说明：

user：未知用户的情况下为空，已知则填用户号码，wifi 下必须填写号码

kServerAddress:DM 服务器地址

kCountDownTime:请求超时时间

## 5.2 发起自动配置

[[RcsCpManager sharedInstance] startCp];

## 5.3 回调

[[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(cpStateChanged:) name:MtcCpDelegateStateChangedNotification object:nil];

```
- (void)cpStateChanged:(NSNotification *)notification {
// 当前自动配置的状态
MtcCpDelegateState state = [[notification.userInfo objectForKey:MtcCpDelegateStatKey]integerValue];
// 此次自动配置的标识
NSNumber *cpId = (NSNumber *)[notification.userInfo objectForKey:MtcCpDelegateCpIdKey];

Switch (state) {
case MtcCpDelegateStatePromptMSISDN:
// PGW 不支持头域丰富，需要通过验证码方式自动配置，提示用户手动输入号码
break;
case MtcCpDelegateStatePromptOTP:
// 正在获取验证码，此时正常流程发起配置的号码会收到短信验证码，更新 manager 的 cpId
[RcsCpManager sharedInstance].cpId = cpId.unsignedIntValue;
break;
case MtcCpDelegateStateCpOk:
// 自动配置流程已完成，此时可以发起注册
break;
case MtcCpDelegateStateCpFailed:
// 自动配置失败
NSNumber *number = (NSNumber *)[notification.userInfo objectForKey:MtcCpDelegateStatCodeKey];
ZUINT statCode = number.unsignedIntValue;
```

```
// 错误码
    switch (statCode) {
        case MTC_CP_STAT_ERR_INVALID_NUMBER:
            failedText = @"无效用户号码，请重新输入";
            break;
        case MTC_CP_STAT_ERR_INVALID_OTP:
            failedText = @"无效验证码，请重新输入";
            break;
        case MTC_CP_STAT_ERR_TIMEOUT:
            failedText = @"请求超时";
            break;
        case MTC_CP_STAT_ERR_NETWORK:
            failedText = @"网络异常";
            break;
        case MTC_CP_STAT_ERR_FORBIDDEN:
            break;
        case MTC_CP_STAT_ERR_INVALID_TOKEN:
            failedText = @"无效 TOKEN，请重新输入";
            break;
        case MTC_CP_STAT_ERR_INTERNAL_ERR:
            failedText = @"服务器内部 500 错误";
            break;
        case MTC_CP_STAT_ERR_INCORRET_XML:
            failedText = @"错误的 XML 配置文件";
break;
}
case MtcCpDelegateStateRecvMsg：
// 收到协议条款，更新 manager 的 cpId，并展示协议内容
            NSNumber *cpId = (NSNumber *)[notification.userInfo objectForKey:MtcCpDelegateCpIdKey];
            NSString *title = [notification.userInfo objectForKey:MtcCpDelegateCpTitleKey];
            NSString *message = [notification.userInfo objectForKey:MtcCpDelegateCpMessageKey];
            [RcsCpManager sharedInstance].cpId = cpId.unsignedIntValue;
}
}
```

## 5.4 验证 OTP 验证码

```
[[RcsCpManager sharedInstance] promptOTPWithCode:self.varifyCodeField.text];
```

参数说明： 手动输入的验证码

## 5.5 协议处理

拒绝：`[[RcsCpManager sharedInstance] rejectCp];`

接受： [[RcsCpManager sharedInstance] acceptCp];