iOS SDK 使用说明(SIP Version)

1. SDK 调用示例说明

```
//定义 CCallbackInterface ,并设置相应回调函数，具体回到的定义第三条详细介绍
CCallbackInterface interface;
interface.onCallProceeding = onCallProceeding;
interface.onCallAlerting = onCallAlerting;
interface.onCallAnswered = onCallAnswered;
interface.onSipConnect = onSipConnect;
interface.onSipLogOut = onSipLogOut;
interface.onIncomingCallReceived = onIncomingCallReceived;
interface.onCallPaused = onCallPaused;
interface.onCallResumed = onCallResumed;
interface.onCallTransfered = onCallTransfered;
interface.onCallReleased =  onCallReleased;
interface.onMeetingTransfered = onMeetingTransfered;
...

// 初始化sdk，传入上面定义的CCallbackInterface，下面每一步的状态都可以通过回调获得
int  ret = servicecoreInitialize(&interface);
// enable tls
ret = setSipTransportType(2);
// srtp
setSrtpEnabled(1, 2);



// ------------------------------ 按钮按下 处理逻辑 ---------------------------  //
if(btn == self.login_in_btn) {
        [self show_communication_state:[NSString stringWithFormat:@"正在连接服务器：%@", [_login_ip text]]];
        // 登录
        int ret = 1;

        ret = setServerAddress(5.2, [[_login_ip text] UTF8String], [[_login_port text] intValue], NULL, 8881, NULL, 1000);
        unsigned int tmp_value = 0;
        ret = connectToCCP(&tmp_value,
                           [[_login_account text] UTF8String],
                           [[_login_password text] UTF8String],
                           "9889",
                           99,
                           "klklk",
                           2, // wifi
                           1,
                           1,
                           "33",
                           "333",
                           "3333",
                           "33333",
                           "333d3d3");
    } else if(btn == self.login_out_btn) {
        // 登出
        [self show_communication_state:[NSString stringWithFormat:@"正在
退出: %@", [_login_ip text]]];
        unsigned int tem = 0;
        disConnectToCCP( &tem );

    } else if(btn == self.call_button_btn) {
        // 呼叫
        [self show_communication_state:[NSString stringWithFormat:@"正在
呼叫: %@", [self.call_num_textfield text]]];

        setUserData(USERDATA_FOR_INVITE, [[self.user_record_textfield
text] UTF8String]);

        char *callId = NULL;
        makeCall((const char**)&callId, VOICE_CALL,
[[self.call_num_textfield text] UTF8String]);
        if(callId)
            memcpy( call_ID, callId, strlen(callId));

        printf("22222正在呼叫:%s\n",call_ID);
         //NSLog(@"正在呼叫:%c",call_ID);
    } else if(btn == self.release_call_btn) {
        // 挂断电话
        [self show_communication_state:@"正在挂断..."];
        releaseCall(call_ID, 0);
    } else if(btn == self.accept_btn) {
        // 接听电话
        int ret = -1;
        printf("22222 acceptCall:%s\n",call_ID);

        setUserData(USERDATA_FOR_200_OK, [[self.user_record_textfield
text] UTF8String]);
        ret = acceptCall(call_ID, 0);
    } else if(btn == self.dtmf_send_button_btn) {
        // 发送DTMF
        //sendDTMF(call_ID, *[[self.dtmf_num text] UTF8String]);
    } else if(btn == self.hands_free_btn) { // 免提
        if(self.hands_free_btn.selected) {
            self.hands_free_btn.selected = NO;
            [btn setTitle:@"免提" forState: UIControlStateNormal];
            enableLoudsSpeaker(false);
            // 开启听筒
        } else {
            self.hands_free_btn.selected = YES;
            // 开启扬声器
            [btn setTitle:@"听筒" forState: UIControlStateNormal];
            if( 0 == enableLoudsSpeaker(true) )
                printf("2222 enableLoudsSpeaker OK \n");
        }

    } else if(btn == self.call_hold_btn) {
        if(self.call_hold_btn.selected) {
            [self show_communication_state:@"呼叫恢复..."];
            self.call_hold_btn.selected = NO;
            [btn setTitle:@"保持" forState: UIControlStateNormal];
            ret = resumeCall(call_ID);

        } else {
            [self show_communication_state:@"保持呼叫..."];
            self.call_hold_btn.selected = YES;
            [btn setTitle:@"恢复" forState: UIControlStateNormal];
            ret = pauseCall(call_ID);

        }
        // 保持
    } else if(btn == self.call_blind_transfer_btn) { //盲转
        [self show_communication_state:@"盲转..."];
        isTransfer = true;
        ret = pauseCall(call_ID);

    } else if(btn == self.call_advice_btn) { //咨询
        [self show_communication_state:[NSString stringWithFormat:@"请主
叫等待,正在咨询%@...", [self.transfer_num text]]];
        char *callid = NULL;
        isAdvice = true;
        ret = pauseCall(call_ID);


    } else if(btn == self.call_advice_release_btn) { //咨询挂断
        // 挂断电话
        [self show_communication_state:@"正在咨询挂断..."];
        releaseCall(call_ID2, 0);


    } else if(btn == self.call_advice_transfer_btn) { //咨询转
        isAdviceTransfer = true;
        [self show_communication_state:@"咨询转..."];
        //releaseCall(call_ID2, 0);
        consultTransferCall(call_ID, call_ID2, [[self.transfer_num
text] UTF8String] );

    } else if(btn == self.call_advice_change_btn) { //咨询切换

        if( isAdviceChange ){
            [self show_communication_state:@"咨询切换到主叫"];
            ret = pauseCall(call_ID2);
            ret = resumeCall(call_ID);
            isAdviceChange = false;
        }else{
            [self show_communication_state:@"咨询切换到第三方"];
            ret = pauseCall(call_ID);
            ret = resumeCall(call_ID2);
            isAdviceChange = true;
        }

    } else if(btn == self.call_advice_meeting_btn) { // 咨询会议
        [self show_communication_state:@"开始咨询会议..."];
        isTransferMeeting = true;
        transferMeeting( CCP_MEETING_TYPE_CONSULT_TRANSFER, call_ID,
call_ID2, NULL );

    } else if(btn == self.call_onemeeting_btn) { // 单步会议

        [self show_communication_state:@"开始单步会议..."];
        transferMeeting( CCP_MEETING_TYPE_SINGLE_STEP, call_ID, NULL,
[[self.dtmf_num text] UTF8String] );

    }else if(btn == self.call_quitmeeting_btn) {//退出会议
        [self show_communication_state:@"退出会议"];
        if( strlen(call_ID) > 0 )
            releaseCall(call_ID, 0);
        if( strlen(call_ID2) > 0 )
            releaseCall(call_ID2, 0);
    }
```

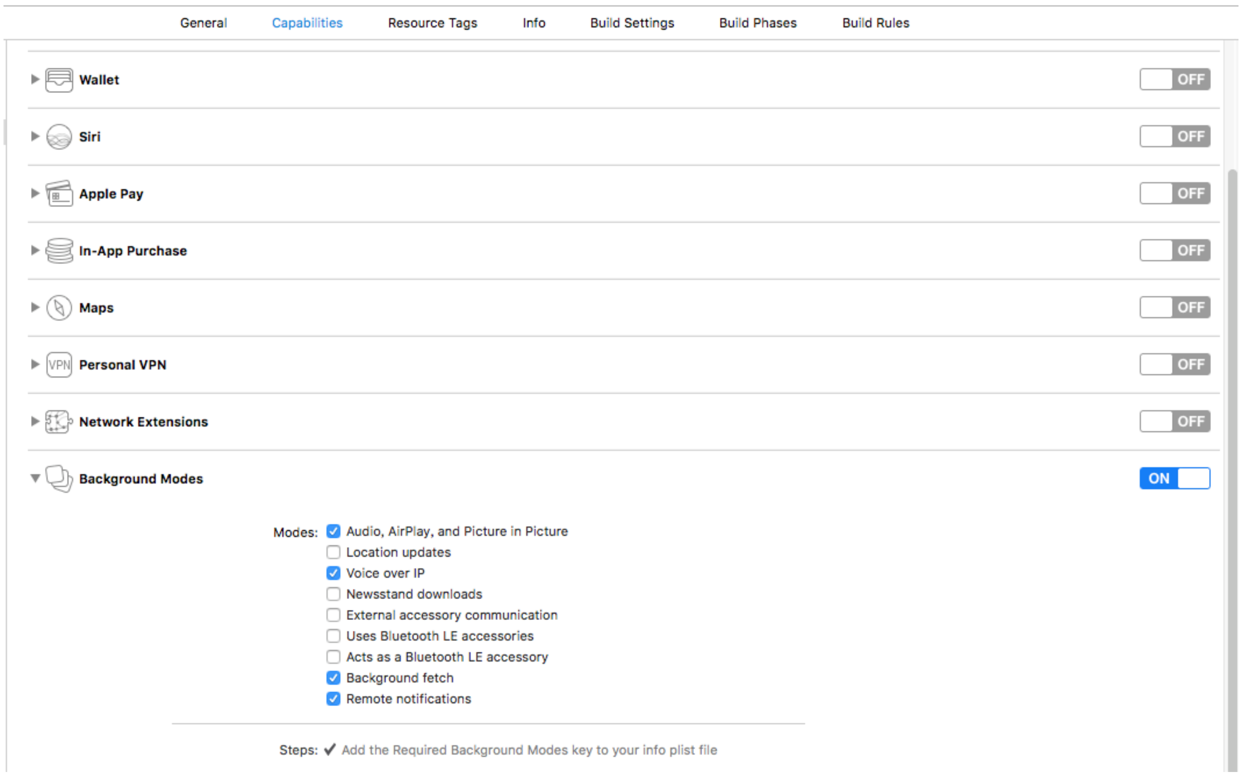
事件回调处理逻辑:

```
void onSipConnect(int reason)
{
    if(reason == 200) {
        [self_handle show_communication_state:@"服务器登录成功!"];
    } else {
        [self_handle show_communication_state:@"服务器登录失败!"];
    }
}
```

```
void onSipLogOut(int reason)
{
    if(reason == 200) {
        [self_handle show_communication_state:@"退出成功!"];
    } else {
        [self_handle show_communication_state:@"退出失败!"];
    }
}
//呼叫已经被云通讯平台处理
void onCallProceeding(const char*callid) {
    [self_handle show_communication_state:@"呼叫处理中..."];
}


//呼叫振铃
void onCallAlerting(const char *callid) {
    [self_handle show_communication_state:@"正在振铃..."];
}
//应答
void onCallAnswered(const char *callid) {
    [self_handle show_communication_state:@"呼叫接通"];
}
void onCallReleased(const char *callid,int reason,int state,int CallEvent)
{
    if( 0 == strcmp(call_ID2, callid ) ){
        memset(call_ID2, 0, sizeof(call_ID2));
        if( isTransferMeeting ){
            isTransferMeeting = false;
            [self_handle show_communication_state:@"已退出咨询会议"];
            return;
        }else if( isAdviceTransfer ){
            isAdviceTransfer = false;
            return;
        }
        [self_handle show_communication_state:@"咨询已挂断,取回主叫通话..."];
        printf("22222 bye the third person OK: callid[%s], reason[%d],
state[%d]", callid, reason, state);
        resumeCall(call_ID);

    }else if( 0 == strcmp(call_ID, callid ) ){
        memset(call_ID, 0, sizeof(call_ID));
        if( isTransferMeeting ){
            printf("22222 call bye: callid[%s], reason[%d], state[%d]",
callid, reason, state);
            return;
        }
        [self_handle show_communication_state:@"已挂断"];
        printf("22222 call bye: callid[%s], reason[%d], state[%d]", callid,
reason, state);
```

```
    }
}
// 呼叫进入
void onIncomingCallReceived(int callType, int confType,const char *callid,
const char *caller) {
    memcpy(call_ID, callid, strlen(callid));
    alertingCall(call_ID);  // alterting, must invoke afer incoming call.
    [self_handle show_communication_state:[NSString stringWithFormat:@"有电
话呼入: %s", caller]];
    //创建一个本地推送
    UILocalNotification *noti = [[UILocalNotification alloc] init] ;
    if (noti)
    {
        //设置时间
        NSDate *date = [NSDate dateWithTimeIntervalSinceNow:1];
        //设置推送时间
        noti.fireDate = date;
        //设置时区
        noti.timeZone = [NSTimeZone defaultTimeZone];
        //设置重复间隔
        noti.repeatInterval = 0;
        //推送声音
        noti.soundName = @"incomingRing.wav";
        NSDictionary *infoDic = [NSDictionary dictionaryWithObjectsAndKeys:
                                 @"value1", @"key1",
                                 @"value2", @"key2",
                                 @"value3", @"key3",
                                 @"value4", @"key4",
                                 nil];
        noti.userInfo = infoDic;

        //添加推送到uiapplication
        UIApplication *app = [UIApplication sharedApplication];
        [app scheduleLocalNotification:noti];
    }

}

//通话保持
void onCallPaused(const char* callid,int type,int reason)
{
    if(reason == 200) {
        [self_handle show_communication_state:@"通话保持成功"];
    } else {
        [self_handle show_communication_state:@"通话保持失败!"];
    }

    if( isAdvice && reason == 200){
        [self_handle show_communication_state:[NSString
stringWithFormat:@"正在咨询%@...", [self_handle.transfer_num text]]];
        //sleep(3);
        makeCall((const char**)&callid, VOICE_CALL,
[[self_handle.transfer_num text] UTF8String]);
        if(callid){
            memcpy(call_ID2, callid, strlen(callid));
        }
        isAdvice = false;
        printf("2222 transfer makeCall:%s\n",call_ID2);

    }else if(isTransfer && reason == 200){
        transferCall(call_ID, [[self_handle.transfer_num text] UTF8String],
0);
        isTransfer = false;

    }
}

//恢复通话
void onCallResumed(const char* callid,int type,int reason)
{
    if(reason == 200) {
        [self_handle show_communication_state:@"恢复通话成功"];
//        if( isAdviceTransfer ){
//            [self_handle show_communication_state:@"开始转接..."];
//            isAdviceTransfer = false;
//            transferCall(call_ID, [[self_handle.transfer_num text]
UTF8String], 0);
//        }
    } else {
        [self_handle show_communication_state:@"恢复通话失败!"];
    }
}
//转接
void onCallTransfered(const char *callid, const char *destionation, int
reason)
{
    if(reason == 200) {
        [self_handle show_communication_state:@"转接成功"];
    } else {
        [self_handle show_communication_state:@"转接失败!"];
    }
}

void  onMeetingTransfered(const char *callid , int reason)
{
    if(reason == 200) {
        [self_handle show_communication_state:@"会议成功"];
    } else {
        [self_handle show_communication_state:@"会议失败!"];
    }
}
```

- App 的background Modes需要做如下设置



General Capabilities Resource Tags Info Build Settings Build Phases Build Rules

Wallet OFF
Siri OFF
Apple Pay OFF
In-App Purchase OFF
Maps OFF
Personal VPN OFF
Network Extensions OFF
Background Modes ON

Modes:
- [x] Audio, AirPlay, and Picture in Picture
- [ ] Location updates
- [x] Voice over IP
- [ ] Newsstand downloads
- [ ] External accessory communication
- [ ] Uses Bluetooth LE accessories
- [ ] Acts as a Bluetooth LE accessory
- [x] Background fetch
- [x] Remote notifications

Steps: ✓ Add the Required Background Modes key to your info plist file

- 关于自动系统电话自动恢复：

  `[[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(resume:) name:UIApplicationDidBecomeActiveNotification object:nil];` 订阅 `UIApplicationDidBecomeActiveNotification`，在函数里调用 `resumeAudio((char*)callid)` 函数（具体定义在下面列表中给出）；

目前为止即为实现简单的通话调用流程，具体接口列表和说明在下面列出。

2. SDK接口列表及使用说明：

```
  /*函数名    : connectToCCPServer
     功能       :登录服务器。防火墙需要开通TCP上行服务器端口8085[连接服务器],8888[下载
服务器],8090[上传服务器]; UDP所有端口
     参数
      [IN]  proxy_addr        : 登录 IP
      [IN]  proxy_port           : 登录 Port
      [IN]  account           : 登录 account
      [IN]  password
      */
     CCPAPI int connectToCCPServer(const char *proxy_addr, int proxy_port,
 const char *account, const char *password);

     /*! @function

 ***************************************************************************
 *****
     函数名    : makeCall
     功能      : 发起呼叫
     参数      :    [IN]  callType    : 呼叫类型enum {VOICE_CALL,//语音VoIP电话
 VIDEO_CALL,//视频VoIP电话   VOICE_CALL_LANDING//语音落地电话};
                 [IN]  called      : 被叫方号码。根据呼叫类型不同，格式也不同
                 VOICE_CALL_LANDING，目前支持只国内呼叫，手机号或者带区号的固话号码，
 例如：13912345678或者01088888888;

                 VOICE_CALL,
                 VIDEO_CALL,
     返回值    : 返回值为callid,本次呼叫的唯一标识; NULL表示失败.
     回调函数 : void (*onCallProceeding)(const char*callid);      //呼叫已经被云
 通讯平台处理
                void (*onCallAlerting)(const char *callid);          //呼叫振
 铃
                void (*onCallAnswered)(const char *callid);          //应答
                void (*onCallReleased)(const char *callid);      //呼叫失败

 ***************************************************************************
 ****/
     CCPAPI int STDCALL makeCall(const char **OutCallid,int callType, const
 char *called);

     /*! @function

 ***************************************************************************
 *****
     函数名    : acceptCall
     功能      : 应答呼入。可以选择媒体类型
     参数      : [IN]  callid     : 当前呼叫的唯一标识
                [IN]  type      : 备用，目前此参数无效
     返回值    : 是否成功 0: 成功;  非0失败
     回调函数 : void (*onCallAnswered)(const char *callid);           //应答

 ***************************************************************************
 ****/
     CCPAPI int STDCALL acceptCall(const char *callid, int type);

     /*! @function
```

```
*****************************************************************************
*****
    函数名   : alertingCall
    功能     : 振铃
    参数     : [IN]  callid     : 当前呼叫的唯一标识
    返回值   : 是否成功 0: 成功; 非0失败
    回调函数 : void (*onCallAnswered)(const char *callid);           //应答

*****************************************************************************
****/
    CCPAPI int STDCALL alertingCall(const char *callid);

    /*! @function

*****************************************************************************
*****
    函数名   : releaseCall
    功能     : 挂机。二十秒没有语音流，SDK自动挂机
    参数     : [IN]  callid     : 当前呼叫的唯一标识,  如果callid 为NULL,这代表所
有呼叫.
               [IN]  reason    : 释放呼叫的原因
    返回值   : 是否成功 0: 成功;  非0失败
    回调函数 : void (*onCallReleased)(const char *callid);

*****************************************************************************
****/
    CCPAPI int STDCALL releaseCall(const char *callid , int reason);
    /*! @function

*****************************************************************************
*****
    函数名   : pauseCall
    功能     : 暂停呼叫，呼叫暂停以后，本地的语音数据将不再传递到对方.
    参数     : [IN]  callid     : 当前呼叫的唯一标识
    返回值   : 是否成功 0: 成功;  非0失败
    回调函数 :   void (*onCallPaused)(const char* callid,int type,int
reason);

*****************************************************************************
****/
    CCPAPI int STDCALL pauseCall(const char *callid);

    /*! @function

*****************************************************************************
*****
    函数名   : resumeCall
    功能     : 恢复暂停的呼叫
    参数      : [IN]  callid      : 当前呼叫的唯一标识
    返回值    : 是否成功 0: 成功; 非0失败
    回调函数 : void (*onResumed)(const char* callid,int type,int reason);

****************************************************************************
****/
    CCPAPI int STDCALL resumeCall(const char *callid);

    /*! @function

****************************************************************************
*****
    函数名    : transferCall
    功能      : 呼叫转移。不支持P2P网络的voip电话呼转
    参数      : [IN]  callid            : 当前呼叫的唯一标识
                 [IN]  destination     : 目标号码
                 [IN]  type            : 呼转类型 (预留)
    返回值    : 是否成功 0: 成功; 非0失败
    回调函数 : void (*onCallTransfered)(const char *callid , const char
*destionation,int reason); //呼叫被转接

****************************************************************************
****/
    CCPAPI int STDCALL transferCall(const char *callid , const char
*destination, int type);

    /*! @function

****************************************************************************
*****
    函数名    : transferMeeting
    功能      : 呼叫转移。不支持P2P网络的voip电话呼转
    参数      : [IN]  type            : 转会议类型。0单步会议
CCP_MEETING_TYPE_SINGLE_STEP, 1 咨询会议CCP_MEETING_TYPE_CONSULT_TRANSFER
                 [IN]  callid          : 当前呼叫的唯一标识。第一路通话
                 [IN]  consultedCallid: 当前呼叫的唯一标识。第二路通话, 及咨询通话;
type==CCP_MEETING_TYPE_CONSULT_TRANSFER时有效,
                 [IN]  consultedUser   : 第三方咨询专家。
type==CCP_MEETING_TYPE_SINGLE_STEP时有效,

    返回值    : 是否成功 0: 成功; 非0失败
    回调函数 : void (*onMeetingTransfered)(const char *callid , int reason);
//呼叫被转会议

****************************************************************************
****/
    CCPAPI int STDCALL transferMeeting(int type,const char *callid, const
char *consultedCallid,const char *consultedUser);
```

```
    /*! @function

************************************************************************************
*****
    函数名    : enableLoudsSpeaker
    功能      : 设置扬声器状态,
    参数      : [IN]  enable : 是否开启
    返回值    : 是否成功 0: 成功;  非0失败

************************************************************************************
****/
    CCPAPI int  STDCALL enableLoudsSpeaker(bool enable);

    /*! @function

************************************************************************************
*****
    函数名    : getLoudsSpeakerStatus
    功能      : 获取当前扬声器否开启状态
    参数      :
    返回值    : true 开启;  false关闭

************************************************************************************
****/
    CCPAPI bool STDCALL getLoudsSpeakerStatus();

    /*! @function

************************************************************************************
*****
    函数名    : setMute
    功能      : 通话过程中设置静音，自己能听到对方的声音，通话对方听不到自己的声音。
    参数      : [IN]  on : 是否开启
    返回值    : 是否成功 0: 成功;  非0失败

************************************************************************************
****/
    CCPAPI int STDCALL setMute(bool on);

    /*! @function

************************************************************************************
*****
    函数名    : getMuteStatus
    功能      : 获取静音状态
    参数      : 无
    返回值    :  true 开启;  false关闭
```

```
*****************************************************************************/
    CCPAPI bool STDCALL getMuteStatus();
```

3. CCallbackInterface 定义：

```
// 定义好回到，传入SDK的初始化函数
typedef struct _CALLBACKINTERFACE CCallbackInterface;

//呼叫回调函数
struct _CALLBACKINTERFACE {
    void (*onIncomingCallReceived)(int callType, int confType,const char *callid, const char *caller);  //接到呼叫 confType：-100 sipcall点对点来电，-1 protobuf点对点来电，大于0 会议来电
    void (*onCallProceeding)(const char*callid);//呼叫已经被云通讯平台处理
    void (*onCallAlerting)(const char *callid); //呼叫振铃
    void (*onCallAnswered)(const char *callid); //进入通话状态(包括主叫和被叫)。主叫接收到这个事件，表明被叫已经应答；被叫接收到这个事件，表明应答成功。
    void (*onCallReleased)(const char *callid,int reason,int state,int CallEvent);  //呼叫挂机。reason：错误码；state:状态值，8外呼等待振铃，9外呼等待应答，当为8或9对应着旧呼叫失败回调;CallEvent：呼叫事件
    void (*onDtmfReceived)(const char *callid, char dtmf);  //收到DTMF按键时的回调
    void (*onCallPaused)(const char* callid,int type,int reason);//通话保持。type，0 本端发起，1对端发起；reason：200成功，其他报错；
    void (*onCallResumed)(const char* callid,int type,int reason);//恢复暂停的通话。type，0 本端发起，1对端发起；reason：200成功，其他报错；
    void (*onMediaDestinationChanged)(const char* callid,int mediaType,const char *ip,int port,int type);//媒体目标地址变化.mediaType 0 音频，1视频；上行目标地址ip和端口port；type=1 点对点,0 服务器中转；
    void (*onNoMicRecording)(const char *callid,int reason);//无麦克采集,没插麦克风报错
    void (*onCallTransfered)(const char *callid, const char *destionation, int reason); //呼叫被转接 reason：202服务器Accepted,200成功，其他失败
    void (*onMeetingTransfered)(const char *callid , int reason); //呼叫被转会议
    /* SIP连接回调
     * reason：100连接中，200成功，403服务器认证失败，其他报错；
     */
    void (*onSipConnect)(int reason);

    /* SIP登出回调。
     * reason：200成功，其他报错；
     */
    void (*onSipLogOut)(int reason);
}
```