# WebRTC SDK API – Ver 3.1

2021/1 2 /23

APP 整合 WebRTC SDK 的規格說明 , 區分3部分 :

----------技術規格說明----------

1. SDK api 介面 ： 分為 call control api 及 media control api 兩大類
2. Callback event : 分為 call control event , media control event 及 system information event 三大類

----------流程示意圖----------

1. Call control 流程示意圖 : connect server , video call , release call ….
2. Media control 流程示意圖 : mute audio/video , switch camera , peerdata ….
3. System event 流程示意圖 : network event , my/peer resolution ….

----------SDK 開發專案整合說明----------

1. iOS 專案整合說明 : 開發環境設定
2. Android 專案整合說明 : 開發環境設定
3. P2P video test 建立P2P視訊測試說明 : 呼叫API與輸入參數說明

# WebRTC SDK API

## Call control API-1

connectMediaServer()：App 撥號前呼叫 , 與 Media Server 建立web-socket連線，並設定Callback Event function
由於iOS 與 Android 存在架構相異性，故**c**onnectMediaServer 呼叫上略有差異，區分如下：

---------- **iOS** ----------

connectMediaServer( string server_wss , event_callback_function event_callback )

App 撥號前呼叫

server_wss ： WebRTC Server Websocket URI . 例如 : "xxx.xxx:4433"

event_callback : SDK callback function for APP 回傳 event.

[註] : APP如果是用page by page模式開發, 當切換page時因為callback function會被清除,所以當切換page時必須再行呼叫
updateCallBackFunction (event_callback_function event_callback )

---------- **Androiod** ----------

connectMediaServer( string server_wss , RTCPhoneSDKObserver event_callback , Context context , EglBase rootEglBase )

App 撥號前呼叫

server_wss : WebRTC Server Websocket URI . 例如 : "xxx.xxx:4433"

event_callback : instance that implement RTCPhoneSDKObserver

context : Context

rootEglBase : EglBase for webRTC video render.

[註] : APP如果是用page by page模式開發, 當切換page時因為callback function會被清除,所以當切換page時必須再行呼叫
updateCallBackFunction (RTCPhoneSDKObserver event_callback )

[註] : connectMediaServer 成功 , App收到callback event : {"event":"server_connect_success","from":"SDK"}

[註] : connectMediaServer 失敗 , App收到callback event : {"event":"server_connect_fail","from":"SDK","cause":"connect timeout"}

# WebRTC SDK API

## Call control API-2

### makeVideoCall() : 建立與Media Server視訊連線,聲音/影像封包開始串流

---

makeVideoCall( string callerid , string calleeid , string module , string userdata)

callerid : 主叫號碼,可自訂( 最多 24 bytes 英數字)

calleeid : 被叫號碼,可自訂( 最多 24 bytes 英數字 )

module : 這通call的執行模組,for 應用區分( 最多 6 bytes 英數字) 例如 : "p2p", "room", "10101"…

userdata : 由APP自行帶入的字串,for 自訂用途( 最多 1024 bytes ) 例如 : {"roomkey":"2181302","token":"digei8…"}

[註] : 撥號成功 , App會收到 : {"event":"call_connect","from":"ms","medium":"video"} ,
表示App已經成功進入模組(流程) .

[註] : 撥號失敗 , App會收到 : {"event":"release","from":"ms","cause":"resource not available"} ,
表示這通call請求失敗 , 原因是"cause":"resource not available"

### makeAudioCall() : 建立與Media Server語音連線,聲音封包開始串流

---

makeAudioCall( string callerid , string calleeid , string module , string userdata)

callerid : 主叫號碼,可自訂( 最多 24 bytes 英數字)

calleeid : 被叫號碼,可自訂( 最多 24 bytes 英數字 )

module : 這通call的執行模組,for 應用區分( 最多 6 bytes 英數字) 例如 : "p2p", "room", "10101"…

userdata : 由APP自行帶入的字串for 自訂用途( 最多 1024 bytes ) 例如 : {"roomkey":"2181302","token":"digei8…"}

[註] : 撥號成功 , App會收到 : {"event":"call_connect","from":"ms","medium":"audio"} ,
表示App已經成功進入模組(流程) .

[註] : 撥號失敗 , App會收到 : {"event":"release","from":"ms","cause":"resource not available"} ,
表示這通call請求失敗 , 原因是"cause":"resource not available"

# WebRTC SDK API

## Call control API-3

releaseCall() : 掛斷與Media Server的視訊或語音連線,包含web-socket連線

---

releaseCall( string cause ) ;

cause : 掛斷原因 : 自行定義字串描述 : 例如: "normal", "reject", "busy"… (參考:附錄A. cause table)

[註] : 對方(或所有與會者)會收到 {"event":"Leave","from":"peer",calleeid:"B","cause":"normal"}

bindLocalView() : 當使用makeVideoCall() 時,App收到 : {"event":"call_connect","medium":"video"}

表示自己的影像串流已經接上, App必須要create顯示自己的view object傳入sdk.

---

bindLocalView( object viewer );

viewer : App顯示自己的 view object

bindRemoteView() : 當使用makeVideoCall() 時,App收到 : {"event":"join","medium":"video","calleeid":"B"}

表示對方的影像串流已經接上, App必須要create顯示對方的view object傳入sdk.

---

bindRemoteView(string calleeid , object viewer );

calleeid : 對方的號碼

viewer : App顯示對方的 view object

# WebRTC SDK API

## Media control API - 1

---

sendDTMF(string dtmf )

發送dtmf按建: ‘0’ , ‘1’ , ‘2’ , ‘3’, ‘4’, ‘5’, ‘6’, ‘7’, ‘8’, ‘9’, ‘*’, ‘0’, ‘#’

---

muteMicrophone( )

麥克風靜音

[註] : sdk 執行完畢會callback : {"event":"Audio_Mute ","from":"sdk"}

sdk 同時會送出 event 給對方 : {"event":"Audio_Mute","from":"peer","calleeid":"A"}

---

unmuteMicrophone( )

取消麥克風靜音

[註] : sdk 執行完畢會callback : {"event":"Audio_UnMute ","from":"sdk"}

sdk 同時會送出 event 給對方 : {"event":"Audio_UnMute","from":"peer","calleeid":"A"}

---

muteCamera( );

遮蔽攝影機(送出黑色畫面)

[註] : sdk 執行完畢會callback : {"event":"Video_Mute ","from":"sdk"}

sdk 同時會送出 event 給對方 : {"event":"Video_Mute","from":"peer","calleeid":"A"}

---

unmuteCamera( );

取消遮蔽攝影機

[註] : sdk 執行完畢會callback : {"event":"Video_UnMute ","from":"sdk"}

sdk 同時會送出 event 給對方 : {"event":"Video_UnMute","from":"peer","calleeid":"A"}

# WebRTC SDK API

## Media control API - 3

---

switchCamera( string camera_device );

切換攝影機前／後鏡頭／螢幕分享

camera_device : 攝影機前/後鏡頭 : “front”: 前鏡頭 ; “rear”: 後鏡頭 ; “screen”: 手機螢幕分享

[註] : sdk 執行完畢會callback : {"event":"Video_Switch_Front","from":"sdk"}

sdk 同時會送出 event 給對方 : {"event":"Video_Switch_Front","from":"peer","calleeid":"A"}

{"event":"Video_Switch_Rear","from":"peer","calleeid":"A"}

---

setAudioOutput( string outputdevice)

設定聲音輸出模式

outputdevice : 聲音輸出模式 : "earphone": 耳機 ; "speaker": 喇叭

[註] : sdk 執行完畢會callback : { "event":"Audio_Earphone","from":"sdk"}

[註] : default : audiocall = "earphone" , videocall = "speaker"

---

sendPeerData (string calleeid , string user-defined-json-data );

傳送自訂的json資料給對方( 透過 webrtc 傳輸管道 ) ;

calleeid : 傳送對象 callerid ; 如果 calleeid=0 表示廣播給所有與會者

user-defined-json-data : 傳送資料內容 , 採json格式 , 欄位內容自訂 (max 512 bytes)

[註] : 對方App會收到 { "event":"peerdata","from":"peer","calleeid":"A","data":{… user-defined-json-data ..} }

[註] : APP 已有送訊管道,不一定要使用這種方式,不過,peerdata優點是:快速,穩定,因為通道web-socket已經建好,

但缺點也是因為必須先建立通道.所以只有在通話過程中才能使用.

# SDK Callback Event

## Callback Event Json 格式 :

```
{ "event":"xxx", "from":"yyy",
  "calleeid":"pp", "roomkey":"{cc}" "data":"{zz}", "value":"vv", "cause":"{ss}",  "medium":"video", "command":"mm"}
```

參數 (mandatory)

- event : 事件字串(不區分大小寫) , 例如 :  call_release, dtls_connect , Join , Leave …..
- from : 事件來源 , 例如 : cm (call-manager) , ms (media-server) , sdk (sdk-callback) , peer (對方)

參數 (optional)

- calleeid : 對方callerid, 當 source=peer 時 , 附帶此參數
- roomkey : 這通call的roomkey,由MS產生(或由makeVideoCall userdata帶入) , 當event==dtls_connect時 , 附帶此參數
- data : 這個事件附屬資料內容
- value : 這個事件附屬數據內容
- cause : 掛斷(離開)原因 , 當event=release時 , 附帶此參數
- medium : 分2種: "audio"=語音 , "video"=影像語音
  當event=Join or dtls_connect 時 , 附帶此參數, 代表對方連上來的medium
- command : 指令內容 ,當event=command時 , 附帶此參數, 代表對方所下的指令

# SDK Callback Event

## Call control Event

release :
// 線路掛斷 , 欄位： event ; source , cause
{ "event":"release","from":"cm","cause ":"APP server reject" }
{ "event":"release","from":"ms","cause ":"Resource not available"}
{ "event":"release","from":"sdk","cause ":"DTLS timeout "}

dtls_connect
// 媒體通道建立完成 , 欄位： event , source , roomkey , medium
{ "event":"dtls_connect","from":"ms","roomkey":"18710102","medium":"audio"}
[註]: medium分2種: "audio"=語音 , "video"=影像語音
App 應對 "medium":"video"呼叫 bindLocalView( )

join
// 對方連上， 欄位： event , source , calleeid , medium
{ "event":"Join","from":"peer","calleeid":"A","medium":"audio"}
[註]: medium分3種: "audio"=語音 , "video"=影像語音 , "cast"=直播觀眾
App 應對 "medium":"video"呼叫 bindRemoteView( )

leave
// 對方離開， 欄位： event , source , calleeid
{ "event":"Leave","from":"peer","calleeid":"A"}

peerdata
// 對方資料訊息， 欄位： event , source , calleeid , data
{ "event":"peerdata","from":"peer","calleeid":"A","data":"…..user-defined-json-data…."}

# SDK Callback Event

## Media control Event - 1

Audio_Earphone

// 聲音輸出設為耳機

```
{ "event":"Audio_Earphone ","from":"sdk"}
```

Audio_Speaker

// 聲音輸出設為喇叭(擴音)

```
{ "event":"Audio_Speaker ","from":"sdk"}
```

Audio_Mute

// 麥克風靜音

```
{ "event":"Audio_Mute ","from":"sdk"}
```

// 對方麥克風靜音

```
{ "event":"Audio_Mute ","from":"peer","calleeid":"A"}
```

Audio_UnMute

// 麥克風解除靜音

```
{ "event":"Audio_UnMute ","from":"sdk"}
```

// 對方麥克風解除靜音

```
{ "event":"Audio_UnMute ","from":"peer","calleeid":"A"}
```

Video_Mute

// 攝影機影像遮蔽

```
{ "event":"Video_Mute","from":"sdk"}
```

// 對方攝影機影像遮蔽

```
{ "event":"Video_Mute ","from":"peer","calleeid":"A"}
```

Video_UnMute

// 攝影機影像解除遮蔽

```
{ "event":"Video_UnMute","from":"sdk"}
```

// 對方攝影機影像解除遮蔽

```
{ "event":"Video_UnMute ","from":"peer","calleeid":"A"}
```

# SDK Callback Event

## Media control Event - 2

Video_Switch_Front

// 切換攝影機: 前鏡頭

```
{ "event":"Video_Switch_Front","from":"sdk"}
```

// 對方切換攝影機: 前鏡頭

```
{ "event":"Video_Switch_Front ","from":"peer","calleeid":"A"}
```

Video_Switch_Rear

// 切換攝影機: 後鏡頭

```
{ "event":"Video_Switch_Rear","from":"sdk"}
```

// 對方切換攝影機: 後鏡頭

```
{ "event":"Video_Switch_Rear ","from":"peer","calleeid":"A"}
```

Video_Switch_Screen

// 切換攝影機: 螢幕分享

```
{ “event”:“Video_Switch_Screen","from":"sdk"}
```

// 對方切換攝影機: 螢幕分享

```
{ "event":"Video_Switch_Screen ","from":"peer","calleeid":"A"}
```

# WebRTC SDK Event

## Syetem Information Event - 1

message

// 系統發送的訊息

{ "event":"message","from":"ms","data":"系統維護中."}

{ "event":"message","from":"ms","data":"所有線路忙線中,請稍後再使用,謝謝."}

{ "event":"message","from":"ms","data":"系統偵測網路狀態不穩"}

{ "event":"message","from":"ms","data":"系統偵測網路延遲"}

{ "event":"message","from":"ms","data":"系統偵測到您的設備沒有送出聲音封包,請檢查麥克風設備"}

{ "event":"message","from":"ms","data":"系統偵測到對方的設備沒有送出聲音封包"}

{ "event":"message","from":"ms","data":"系統偵測到您的設備沒有送出影像封包,請檢查攝影機設備"}

{ "event":"message","from":"ms","data":"系統偵測到對方的設備沒有送出影像封包"}

my_resolution

// 我的聲音影像,網路品質參數 (約每5秒,送一次)

{ "event":"my_resolution","from":"ms","data":"{ "width":600,"height":480,"video_bitrate":1045,
"REMB":1000,"framerate":30,"audio_bitrate":20,"jitter":54,"packetloss":0 }"}

width:攝影解析度:寬 ； height:攝影解析度:高 ； REMB:目前預設的video biitrate ；video_bitrate:實際video bitrate

framerate:每秒幾張畫面 ；audio_bitrate:實際audio bitrate ；jitter:聲音封包抖動參數(ms)； packetloss:封包遺失數量

[註] : App 可以針對 jitter>1000 或 packetloss>10 , 顯示"我方網路品質不良"訊息.

peer_resolution

// 對方B的聲音影像,網路品質參數 (約每5秒,送一次)

{ "event":"peer_resolution","from":"ms","calleeid":"B","data":"{ "width":600,"height":480,"video_bitrate":1045,
"REMB":1000,"framerate":30,"audio_bitrate":20,"jitter":54,"packetloss":0 }"}

[註] : App 可以針對 jitter>1000 或 packetloss>10 , 顯示"對方網路品質不良"訊息.

# WebRTC SDK Event

## App 通話過程中互送 data Event

peerdata
// 來自對方的訊息
{ "event":"peerdata","from":"peer","calleeid":"1235","data ":{… user-defined-json-data ..} }
[註] : 送方是透過 sdk-api : sendPeerData (string calleeid , string user-defined-json-data );
[註] : user-defined-json-data : 可以由app自行決定欄位內容,只要是符合json格式即可,最大長度限制:512 bytes
[註] : 在多方會議室時,送方可以將calleeid設為"0",表示廣播給所有與會者 : sendPeerData ( "0", {"message":"Hello"} );
[註] : APP app 已有送訊管道,不一定要使用這種方式,不過,peerdata優點是:快速,穩定,因為通道已經建好,
但缺點也是因為必須先建立通道.所以只有在通話過程中才能使用.

## 附錄A. cause table

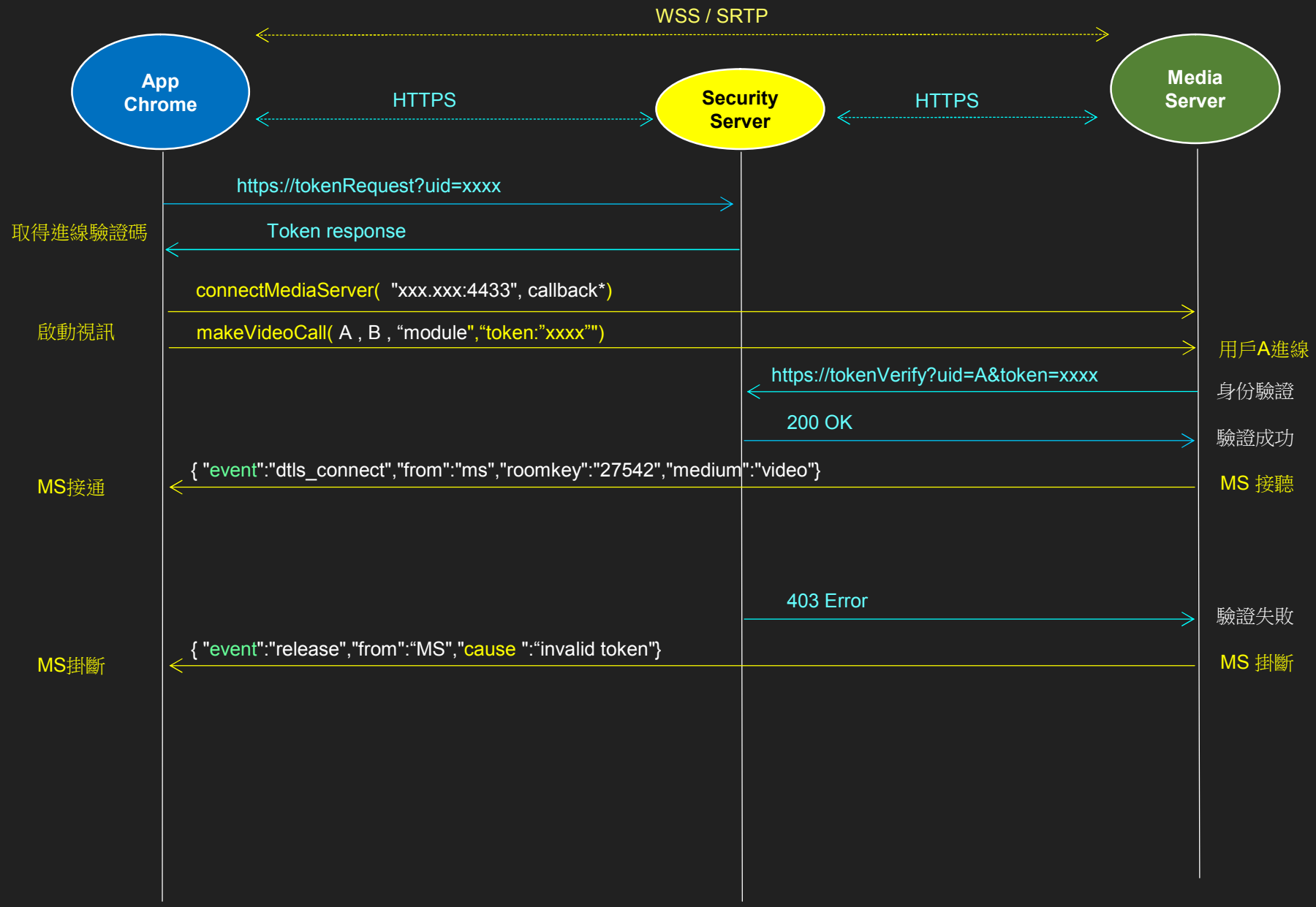
---

**cause** : 掛斷原因 : APP自行定義字串 : 例如: "normal", "reject", "開會中", "忙碌中"…

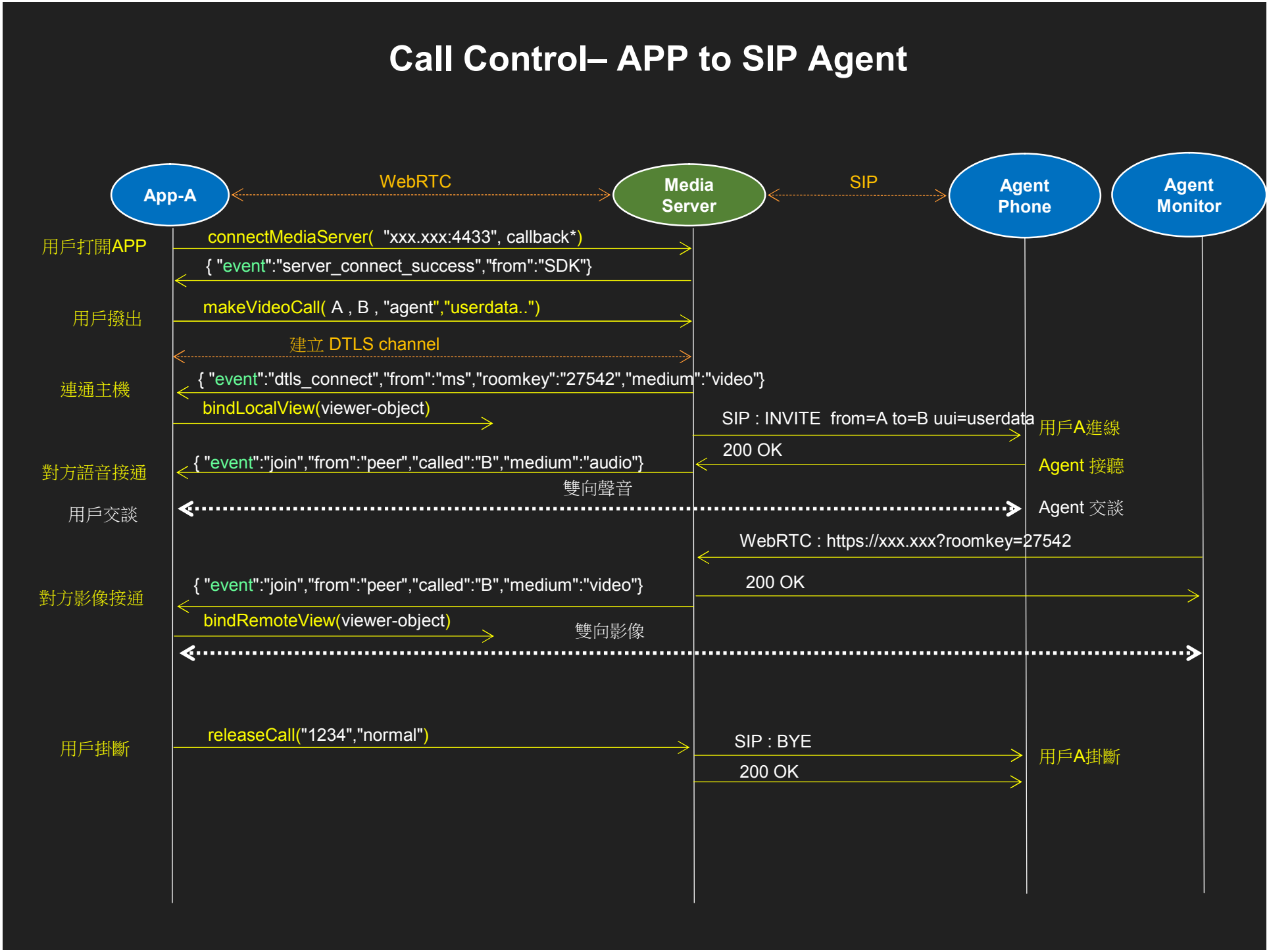
**系統可能發生的cause:**

"normal release" : 正常掛斷
"no circuit available" : 無空餘線路資源
"user busy" : 對方忙線
"no answer" : 對方響鈴未接來電
"call rejected" : 對方拒絕
"request timeout" : 通信timeout
"websocket closed" : websocket網路斷線(data)
"rtpsocket closed" : rtpsocket網路斷線(media)

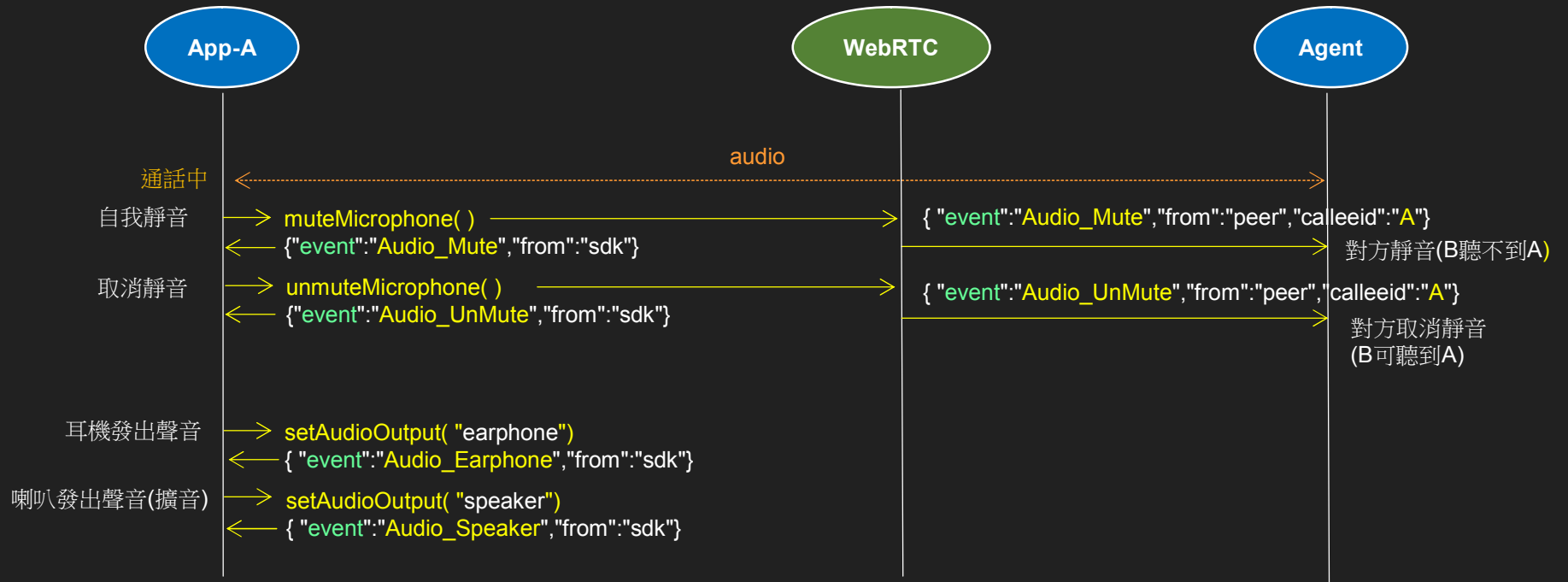
# Security Verify – APP or Browser



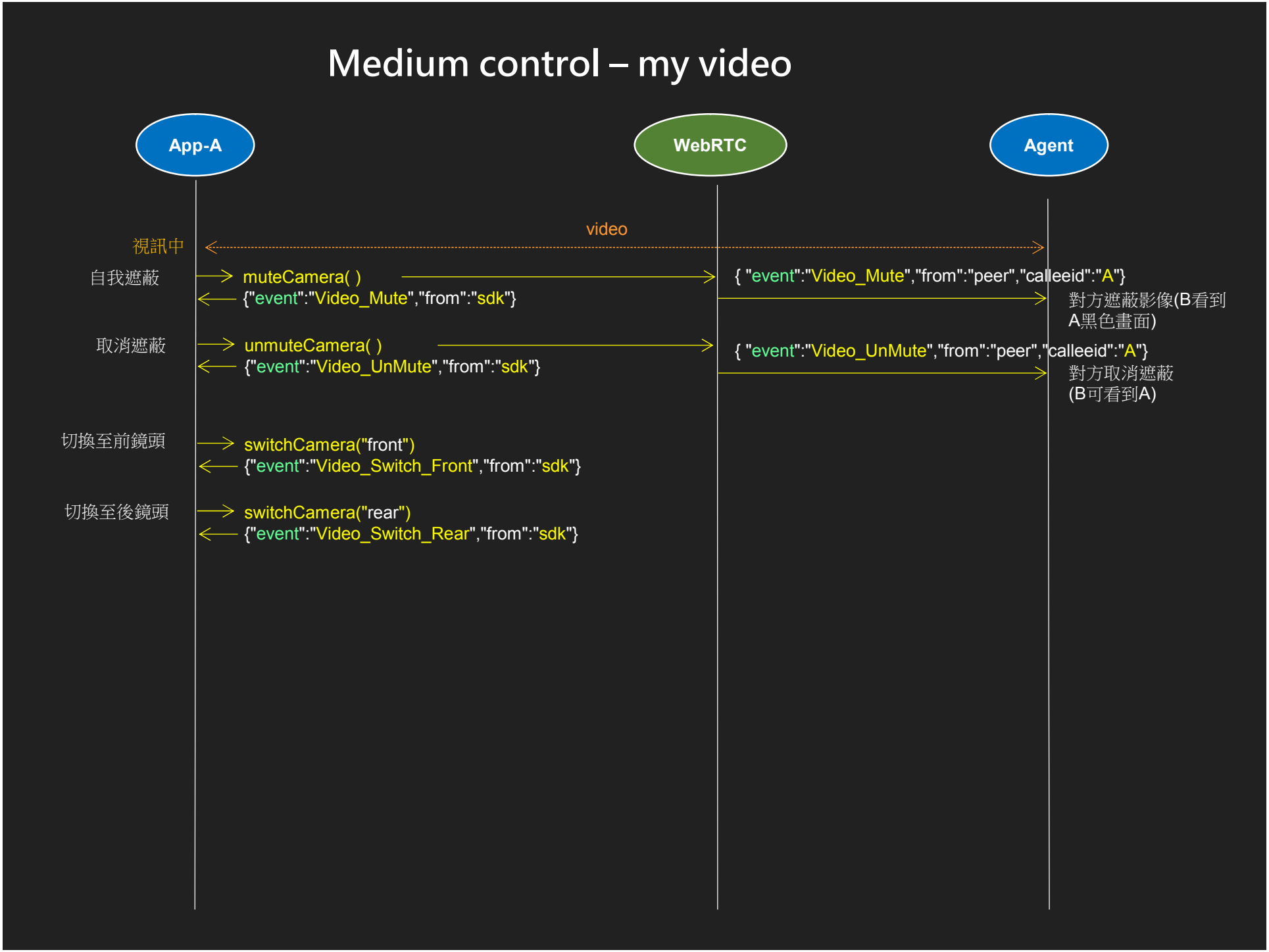
# Call Control– APP to SIP Agent



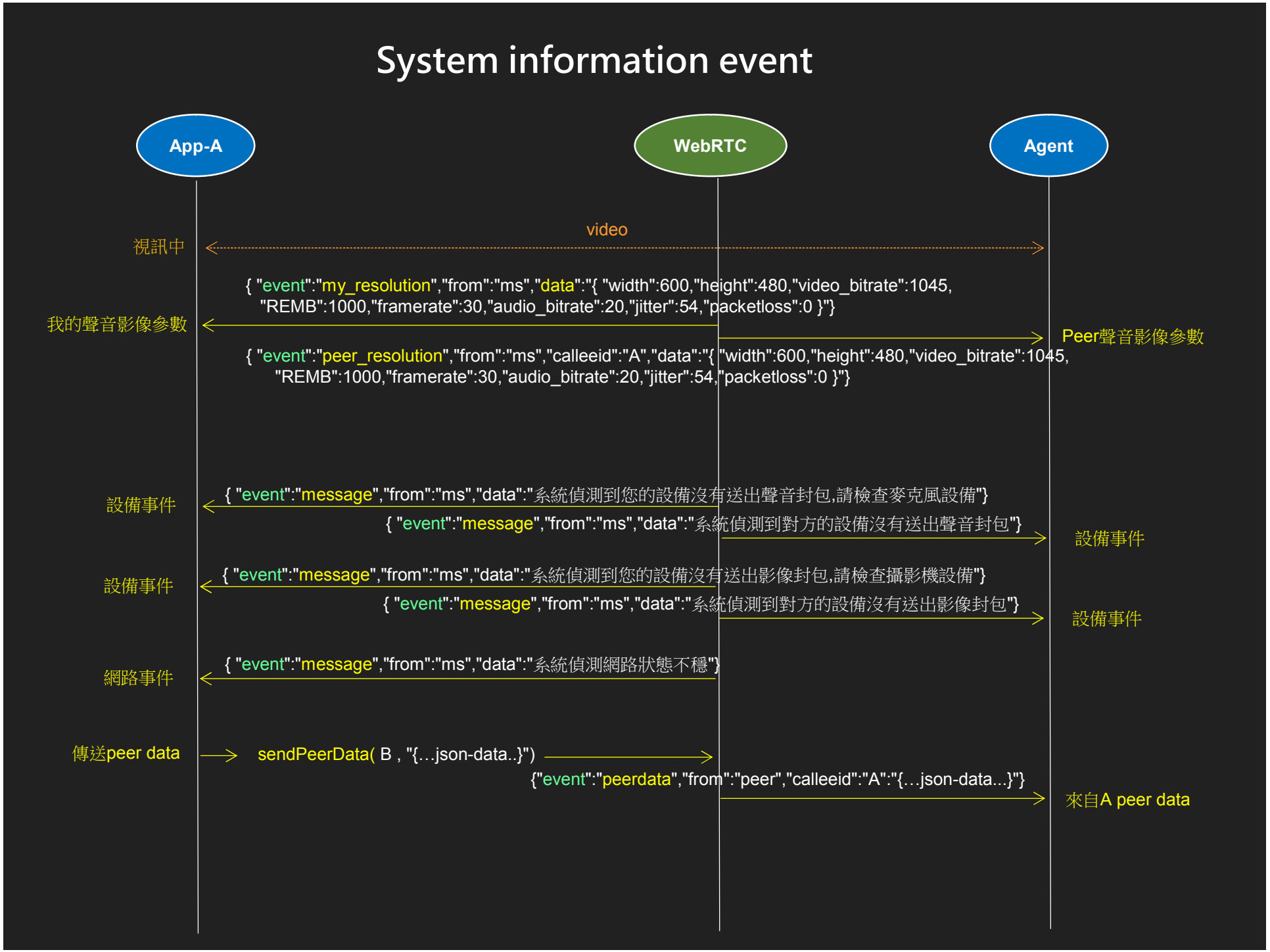
# Medium control – my audio



# Medium control – my video



# System information event



# iOS 專案整合說明

一、檔案說明：PhoneSDK 資料夾下，包含

1、include folder:

(1). library header file

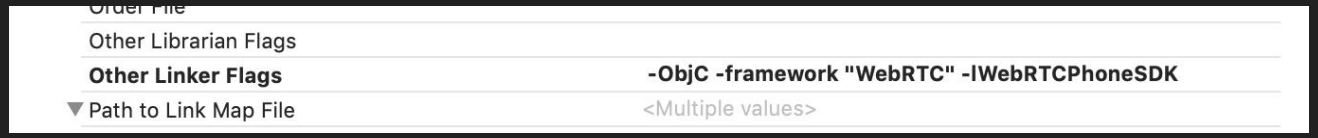
(2). 必要framework : webrtc framework

2、libWebRTCPhoneSDK.a

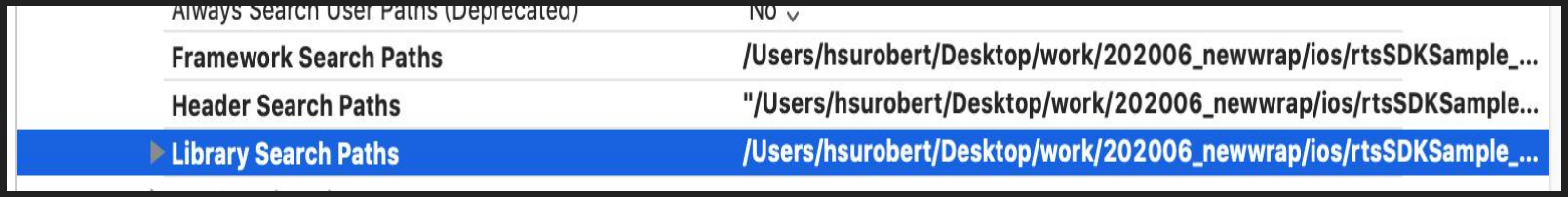
二、環境設定

1、複製PhoneSDK資料夾 到專案目錄

2、設定Other Linker Flags




3、設定 search paths (Framework、Header、Library)



三、開發說明：以Objective-C 為例

1、 #import "WebRTCPhoneSDK.h"

2、makeVideoCall前呼叫 "connectMediaServer"

```
[[WebRTCPhoneSDK sharedInstance] connectMediaServer:self.mstr_PhoneServer callbackevent:^(NSDictionary
    *phoneCallback) {
    NSLog(@"%@",phoneCallback);
    [self handleWebRCTPhoneCallback:phoneCallback];
}];
```

3、 handle callback Event ： callbackevent 為 dictionary

處理概述(詳細流程請參閱上述文件內容)：

(1)、 server_connect_success: 連線成功，可開始建立通話流程

```
if ([event isEqualToString:@"server_connect_success"]){
    if (!self.isVideoCall) {
        [[WebRTCPhoneSDK sharedInstance] makeAudioCall:self.mstr_PhoneCallFrom
            calleeID:self.mstr_PhoneCallTo module:self.mstr_CallModule userdata:self.mstr_UserData];
    }
    else {
        [[WebRTCPhoneSDK sharedInstance] makeVideoCall:self.mstr_PhoneCallFrom
            calleeID:self.mstr_PhoneCallTo module:self.mstr_CallModule userdata:self.mstr_UserData];
    }

}
```

(2)、 收到 event : dtls_connect : 連線建立成功，呼叫 bindLocalView()

```
else if([event isEqualToString:@"dtls_connect"]){//connection finished
    [self enableButtons];
    [self.theCallManager bindLocalView:self.mLocalView];

    if([_strCallMedium.lowercaseString isEqualToString:@"video"] && [[UIAppl
        set speaker
```

(3)、 收到 event : join & medium=video : 表示對方連線，呼叫bindRemoteView 進行視訊

```
else if([event isEqualToString:@"Join"]){//connection finished
    if (self.isVideoCall){
        [[WebRTCPhoneSDK sharedInstance] bindRemoteView:self.mstr_PhoneCallTo removeView:self->remoteVideoView];
    }
}
```

# Android 專案整合說明

一、檔案說明：rtcPhoneSDK.aar 為webrtc phone sdk

二、環境設定

1、設定build.gradle ，新增
implementation 'org.webrtc:google-webrtc:1.0.+‘
implementation "org.java-websocket:Java-WebSocket:1.4.0"

2、設定AndroidManifest.xml 權限

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.rtcphone.rtcphonesdksample">
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.RECORD_AUDIO" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
    <uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
    <uses-permission android:name="android.permission.MANAGE_OWN_CALLS"/>
    <uses-permission android:name="android.permission.READ_CALL_LOG"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"></uses-permission>
```

3、將rtcPhoneSDK.aar複製到專案libs 資料夾

三、開發說明：以Java 為例(詳細流程請參閱上述文件內容)：

1、 import com.rtcphone.rtcphonesdk.*;

2、實作 RTCPhoneSDKObserver

```
@Override
public void RTCPhoneSDKCallback(JSONObject objCallback) {
    Log.d(TAG,  msg: "webRTCCallManagerCallback: "+ objCallback.toString());
```

3、 makeVideoCall 前呼叫 : connectMediaServer()

```
RTCPhoneSDK.getInstance().connectMediaServer(this.strCaller,this.serverURI,this,this,rootEglBase);
```

4、處理RTCPhoneSDKObserver.RTCPhoneSDKCallback

(1)、server_connect_success: 連線成功，可開始建立通話流程

```
else if(event.equalsIgnoreCase( anotherString: "server_connect_success")){
    if(this.isVideoCall()){
        RTCPhoneSDK.getInstance().makeVideoCall(this.strCaller,this.strCallee,this.strModule,this.strMeta);
    }
```

(2)、 收到 event : dtls_connect : 連線建立成功，呼叫 bindLocalView()

```
else if(event.equalsIgnoreCase( anotherString: "dtls_connect")){
    if(this.isVideoCall())
    {
        callingViewActivity.this.currentSound = "earphone";
        btnChangeSpeeaker.callOnClick();
        RTCPhoneSDK.getInstance().bindLocalView(localVideoView);
    }

    this.startCountTime();

}
```

(3)、 收到 event : join & medium=video : 表示對方連線，呼叫bindRemoteView 進行視訊

```
else if(event.equalsIgnoreCase( anotherString: "join")) {
    this.isPeerJoin = true;
    if(objCallback.has( name: "peer_uid")){
        if(this.isVideoCall()) {
            RTCPhoneSDK.getInstance().bindRemoteView(objCallback.getString( name: "peer_uid"), this.remoteVideoView);
        }
    }

}
```

# 建立P2P視訊測試

1. 測試APP API相關參數:
   1. connectMediaServer()

      server_wss : wss://rtc.tw:4433　　[註] : 本公司測試用Media Server主機

   2. makeVideoCall()

      callerid : "1234"　[註] : 本測試流程未使用此參數
      calleeid : "5678"　[註] : 本測試流程未使用此參數
      module : "room"　[註] : 本測試流程固定使用"room"參數
      userdata : {"roomkey":"12345678"}　[註] : 本測試流程只要 roomkey 相同即可配對視訊 .

2. 測試步驟 :
   1. 請準備2支iOS或Android手機,可同時(或先後)執行視訊連線,如果能夠互相視訊,表示已經完成P2P視訊連線. (如果不能視訊,請確認callback event內容)
   2. 視訊過程中,任一方可以掛斷再進入,應該能夠繼續視訊.
   3. 本項測試只有支援P2P雙方視訊(未防呆),第3方請勿採用相同"roomkey"進入.
   4. 本項測試可支援多組P2P視訊,各組只要"roomkey"相同即可.